UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN CIENCIAS MATEMÁTICAS Y
DE LA ESPECIALIZACIÓN EN ESTADÍSTICA APLICADA

INDUCTIVE AND COINDUCTIVE TYPES IN HOMOTOPY TYPE THEORY. AN EXPERIMENT ON THE KNASTER-TARSKI CONSTRUCTION

TESIS
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO (A) EN CIENCIAS

PRESENTA:
JESÚS HÉCTOR DOMÍNGUEZ SÁNCHEZ

DIRECTOR DE TESIS:
DR. FAVIO EZEQUIEL MIRANDA PEREA
FACULTAD DE CIENCIAS

CIUDAD DE MÉXICO, JUNIO DE 2016.

Acknowledgments

I want to thank my advisor Dr. Favio E. Miranda Perea for suggesting Homotopy Type Theory as a dissertation topic. He gave me complete freedom in exploring this exciting area. He also showed a lot of patience when I attempted to explain a concept.

Also, I want to thank all people, present and past, involved in the development of Homotopy Type Theory. They created an awe-inspiring and beautiful theory. I am excited about the "ultimate form" this theory will take in the future. These are truly exciting times to be alive.

I want to dedicate this work to my parents. They have always encouraged and supported my decisions, specially when these decisions involve learning. However, they will probably find amusing that their son has produced another book in the "scribbles category" 😄.

Contents

Introduction

The construction of fixpoints is a common tool in mathematics to justify inductive and coinductive definitions [16]. Among the most common examples are inductively and coinductively defined sets, and recursively defined functions.

In order theory, the Knaster-Tarski Lemma [16] is one of the main tools to define least and greatest fixpoints for monotone functions. Together with the Knaster-Tarski Lemma, there are also iterative approaches for building least and greatest fixpoints, under suitable conditions on the monotone function [16].

It is well-known that many iterative approaches for building fixpoints can be generalized into category-theoretic constructions [8]. These generalizations roughly use the following "translation table":

Order Theory	**Category Theory**
Ordered set	Category
$A \leq B$	Morphism: $f : A \to B$
Bottom element	Initial object
Top element	Final object
Monotone function	Functor
Increasing sequence	Chain
Decreasing sequence	Cochain
Supremum of increasing sequence	Colimit of chain
Infimum of decreasing sequence	Limit of cochain
Prefixpoint	Algebra
Postfixpoint	Coalgebra
Least fixpoint	Initial algebra
Greatest fixpoint	Final coalgebra

Table 1: Concept translation from Order Theory to Category Theory

The categorical generalization has the advantage of building not only the objects serving as least and greatest fixpoints (i.e. an initial algebra and a final coalgebra), but also the principles for constructing morphisms (or functions) out of the initial algebra and into the final coalgebra. These principles are usually called recursion and corecursion principles [17].

Although the literature focuses on carrying out these generalizations into category theory, it is interesting to see how these constructions can be expressed in alternative formalisms, like type theory.

A further motivation is that we are usually interested in "effectively constructing" fixpoints (i.e. how to obtain the fixpoint through an algorithm). Type theories are well-suited for this kind of "constructive thinking".

Type theory started as a mechanism for avoiding paradoxes in set theory, and quickly evolved into a formalism that is powerful enough to express logics, and as a consequence, mathematical theories (see Section 1.1 for more details).

Homotopy Type Theory is a recent upshot of this evolving nature of type theory. This recently developed type theory embodies a very expressive constructive logic. Also, it is currently proposed as a foundation for mathematics, as an alternative to Zermelo-Fraenkel Set Theory [20].

Set theory axiomatizes the undefined notion of *collection*, while type theory focuses on formalizing the undefined notion of *type*. The meaning of types has changed since their invention. Initially, types were classification mechanisms for avoiding paradoxes and excluding unwanted expressions. For example, stating that a function f can be applied on natural numbers excludes expressions like $f(f)$, where f is applied on itself. In other words, any

input to f must have the *type* of natural numbers, or equivalently, any input to f must be *classified* as a natural number.

In a later development, baptized as the Curry-Howard correspondence [18], types were interpreted as logical propositions, opening the door for type theories to encode logics [13]. More recently, Homotopy Type Theory added another interpretation: types are a special kind of topological spaces [20]. The interpretation provided by Homotopy Type Theory even subsumes the logical interpretation as a special case.

A common feature on a type theory (and Homotopy Type Theory is not the exception) is that it introduces inference rules, which express how types can be formed, how expressions can be constructed and assigned a type, and how expressions can be simplified (or how computation is performed on expressions).

Among the many kinds of types that a type theory can define, we will be interested on those called *inductive types* and *coinductive types* (see Sections 1.9 and 1.10).

An inductive type captures the idea that *all* its expressions can be built from the "bottom up", by iteratively applying constructors. For example, to define the type of natural numbers (seen as an inductive type), we state that the symbol 0 is a natural number (i.e. the symbol 0 acts as a base constructor), then we state that if n is a previously constructed natural number, the expression $S(n)$ is also a natural number (i.e. the symbol S acts as a recursively defined constructor).[1] Hence, any natural number can be constructed by repeatedly applying the constructor S on the base constructor 0.

Dually, a coinductive type captures the idea that its expressions can be "decomposed" into an observation part and a next state part. Coinductive types use destructors to decompose expressions, while inductive types use constructors to build up expressions. For example, the coinductive type of infinite lists over natural numbers (also called streams), can be defined by two destructors. The first destructor, denoted by the expression $head(s)$, extracts the first element of stream s (in other words, the symbol *head* is the destructor producing the current available observation in s). The second destructor, denoted by the expression $tail(s)$, removes the first element from stream s (in other words, the symbol *tail* is the destructor producing the next state of s, so that this next state is ready to be decomposed again).

In type theory, there are many alternative ways to define (co)inductive types. One of them consists on using initial algebras and final coalgebras (see [20] and [4]). *Initial algebras encode inductive types and final coalgebras encode coinductive types.* This claim will me made precise in Sections 1.9 and 1.10.

Therefore, it is very important to investigate the construction of initial algebras and final coalgebras. The literature usually focus on iterative approaches. These approaches construct initial and final coalgebras by computing colimits of chains and limits of cochains, see [8] and [1].

The iterative approaches are categorical generalizations (through the "translation table" above) of well-known order-theoretic techniques, which obtain least and greatest fixpoints by computing the least upper bound of increasing sequences and the greatest lower bound of decreasing sequences, see [16].

Hence, if iterative approaches can be generalized, a natural question arises: Is the Knaster-Tarski Lemma (which is an order-theoretic result) susceptible to generalization into the language of algebras and coalgebras? Of course, instead of working in Category Theory, we want to develop our results in Homotopy Type Theory.

We can now state the main goal of this document.

> We will experiment with the question (and explore some of its consequences) of whether or not initial algebras and final coalgebras can be constructed in Homotopy Type Theory by directly translating the proof of the Knaster-Tarski Lemma into the language of algebras and coalgebras.

We will show that the main consequence of such experiment is that the constructed (co)algebras fail to be initial/final *only by a universe level*. These structures will be called *lower level initial algebras* and *lower level final coalgebras*.

Lower level initial/final (co)algebras are interesting, because we will show that *any* endofunctor has a *lower level* initial algebra, and *any* set-preserving[2] endofunctor has a *lower level* final coalgebra. To understand why this claim is interesting, just consider that there are set-preserving endofunctors without initial/final (co)algebras (see Sections 3.2.3 and 3.3.3).

In spite of the universe level restriction, lower level initial/final (co)algebras are still *useful*, because they have (co)iteration principles that allow the definition of *unique* functions by (co)recursive equations.

[1] In standard mathematics, another clause must be added expressing that 0 and S are the only ways in which natural numbers can be constructed. In type theory this clause is omitted, because it is captured by "elimination rules", see Section 1.5.

[2] Preservation of sets is made precise in Section 1.9.

Another consequence is the generality of the results, as endofunctors only need to preserve sets. To the contrary, iterative approaches impose stronger conditions (for example, endofunctors need to preserve colimits of chains and limits of cochains [1]).

In addition to the main results, this report provides two detailed examples on how lower level initial/final (co)algebras are used: the natural numbers seen as a lower level initial algebra, and the streams seen as a lower level final coalgebra.

A secondary goal of this document is to provide more evidence that Homotopy Type Theory is a very powerful formalism for doing mathematics.

Also, the reader will notice that almost all proofs on this report have a lot of detail, even to a level that a mathematician might deem unnecessary. The author made the decision of including very detailed proofs because Homotopy Type Theory is a very recent and relatively unknown formalism, and the only way to build intuition when learning a new formalism is to actually carry out all the details.

On this document we are working within the framework of computer formalized mathematics [22]. As such, another advantage of having detailed proofs is that they can be checked in interactive proof assistants, like Coq [24]. For example, the contents of Chapters 2 and 3 were computer-checked in Coq. See [26] for details on how to download the coq script file containing the results on this document.

We make a final point regarding notation. On this report, it will be common to switch between standard mathematics and Homotopy Type Theory, specially when doing comparisons between formalizations of a particular concept. To differentiate between the two formalisms, we will use the following typographical conventions.

When an expression is written in standard mathematics, we will use italic font:

$$a,\ b,\ c,\ W,\ X,\ A,\ f$$

When an expression is written in Homotopy Type Theory, we will use sans serif font:

$$\mathsf{a},\ \mathsf{b},\ \mathsf{c},\ \mathsf{W},\ \mathsf{X},\ \mathsf{A},\ \mathsf{f}$$

This document is structured as follows:

- Chapter 1 provides a minimal presentation of Homotopy Type Theory. It introduces just the amount of material to be able to prove the results on Chapters 2 and 3.

  The chapter also defines the important concepts of functor, algebra, coalgebra, algebra and coalgebra morphism, initial algebra, final coalgebra, inductive type and coinductive type (see Sections 1.9 and 1.10).

- Chapter 2 formalizes in Homotopy Type Theory all the basic concepts found in order theory, like partially ordered set, monotone function, complete lattice, infimum, supremum, least and greatest fixpoint.

  It also provides a proof in Homotopy Type Theory of the Knaster-Tarski Lemma. The proof of this lemma provides the construction to be generalized in Chapter 3.

  The chapter also constructs induction and coinduction principles as corollaries to the Knaster-Tarski Lemma.

- Chapter 3 contains the main results on this report.

  Section 3.1 provides an introduction to the iterative approach for building initial/final (co)algebras, as usually found in the literature.

  Sections 3.2 and 3.3 carry out the generalization of the the Knaster-Tarski construction for inductive and coinductive types, respectively.

  These sections prove the existence theorems for lower level initial/final (co)algebras. They also provide an example of a set-preserving endofunctor having a lower level initial/final (co)algebra but not an initial/final (co)algebra.

  In addition, these sections show detailed examples on how lower level initial/final (co)algebras are used. They also discuss limitations on the experiment.

- Chapter 4 summarizes the experiment findings and provides a list of questions for future work.

Chapter 1

Homotopy Type Theory (HoTT)

This chapter provides a self-contained presentation for Homotopy Type Theory (HoTT). It is based on the HoTT book [20]. However, it will present only the minimal amount of results that are required for the developments on Chapters 2 and 3.

As such, it should not be taken as a full exposition of HoTT, as there are amazing applications that the author had to leave out. Nevertheless, all basic types and axioms found on the standard presentation are included, so that one can start formalizing mathematics, as Chapters 2 and 3 will show. The reader is invited to have a look at the HoTT book [20] for an extended presentation.

This chapter is structured as follows:

- Section 1.1 provides a short historical account for type theory, and a motivation for the concept of type.
- Section 1.2 presents the syntax and basic concepts required for carrying out formal proofs in HoTT.
- Section 1.3 introduces type universes and contexts.
- Section 1.4 lists structural rules required in formal proofs.
- Section 1.5 introduces all HoTT standard types, together with inference rules for reasoning with them.
- Section 1.6 lists some basic results and introduces the important concepts of equivalence and type equivalence. Also, the function extensionality axiom and the univalence axiom are introduced.
- Section 1.7 provides an informal account of the homotopy (or topological) interpretation for HoTT.
- Section 1.8 introduces the concepts of contractible type, proposition, and set, together with some of their properties. This section also introduces the propositional resizing axiom.
- Section 1.9 presents the important concept of inductive type, which is formalized through the concept of initial algebra. This section also presents the important notions of functor and algebra morphism.
- Section 1.10 presents the important concept of coinductive type, which is formalized through the concept of final coalgebra. This section also presents the important notion of coalgebra morphism.
- Section 1.11 introduces the concept of higher inductive type (HIT), together with some HITs that will be required on later chapters.

1.1 Background on type theory

A detailed historical account for type theory can be found in [13]. This section only provides a very short summary.

The origins of type theory can be traced back to Bertrand Russell's *ramified theory of types* as a way to solve set-theoretic paradoxes at the time.

The need to simplify Russell's ramified theory of types, prompted Frank Ramsey to introduce his *simple theory of types*. This type theory influenced the development of the *simply typed* λ*-calculus*, due to Alonzo Church, which introduced the concept of function as a primitive concept.

The next development occurred with the discovery of the *Curry-Howard correspondence* or *propositions-as-types correspondence* [18], which was independently discovered by many people (Heyting, Kolmogorov, Curry, Feys, Howard). The Curry-Howard correspondence arose as the realization that types can be interpreted as propositions in a logic, and terms in the type theory as proofs for propositions. For example, a proof in propositional logic for the implication $p \to q$ can be interpreted as a typed function in the simply typed λ-calculus.

The Curry-Howard correspondence opened up the unification between logic and type theory, which meant that a type theory could also be used *as a logic*.

In the light of the Curry-Howard correspondence, Martin Löf developed the now called *Martin Löf's Type Theory*, which extends the simply typed λ-calculus with type universes, identity types, and dependent types. In this type theory, dependent types correspond to logical quantifiers in first-order intuitionistic logic.

Homotopy Type Theory is a recent development. It is Martin Löf's Type Theory extended with the univalence axiom (see Section 1.6.2) and higher inductive types (see Section 1.11). The beginnings of HoTT can be traced back to the discovery that Martin-Löf's Type Theory has a topological model ([23] and [21]). In the topological model, types are spaces and proofs for identities are paths in spaces. The univalence axiom was later discovered by Voevodsky [14].

The fact that type theories can be used as a logic made possible the development of proof assistants for the formalization of mathematics. For example, the contents of Chapters 2 and 3 were formalized in the Coq proof assistant [24].[1] Also, many of the results on the current chapter are formalized in the HoTT Coq library [25].

The key concept in a type theory is that of *type*. What is a type? The answer to this question has evolved since types were invented.

One way to look at a type is that it is a syntactical classification mechanism for removing unwanted expressions in a formal language. For example, if we have a function f from natural numbers to natural numbers, then it is an invalid operation to apply this function on rationals, or on matrices, or on itself, etc. The type of the input *classifies* the expressions that can be given as input to f. This is the way in which types are commonly used in programming languages.

Another way to look at types is that they are propositions, due to the Curry-Howard correspondence [18]. Even another way to look at types is that they are sets, where all expressions having the type denote elements in the set. Hence, by mixing the logical interpretation with the set interpretation, a type can also be seen as the set of all proofs of the proposition it denotes.

HoTT introduces even another interpretation: types are spaces [20]. This interpretation has the advantage of clarifying the meaning of many expressions in type theory, specially the behavior of identity types. Also, this interpretation made possible the discovery of the univalence axiom. It even subsumes the set and logical interpretations as special cases (sets and propositions can be represented as a special kind of spaces, see Section 1.8). With all these advantages, HoTT becomes a suitable alternative for the foundations of mathematics.

The following sections are devoted to provide a self-contained presentation of HoTT.

1.2 Syntax, judgments, and derivation trees

First, we define the syntax for HoTT. We assume the existence of a countably infinite set of variables $\mathcal{VAR}$. We will denote variables by letters $\mathsf{v}, \mathsf{w}, \mathsf{y}, \mathsf{z}$, or numerated letters (for example $\mathsf{w1}, \mathsf{w2}, \mathsf{y5}, \ldots$), or longer names if necessary.

There will be two *syntactical classes*[2] in our type theory: syntactical terms and syntactical contexts.

Definition 1.2.1. The set of *syntactical terms* $\mathcal{TERM}$ (or *S-terms* for short) is inductively defined by the following clauses:

- Every variable in $\mathcal{VAR}$ is an S-term.
- If $i \geq 0$ is a natural number, then the symbol $\mathcal{U}_i$ is an S-term called a *type universe at level* i.
- If v is a variable, and T, R are S-terms, then $(\prod_{\mathsf{v}:\mathsf{T}} \mathsf{R})$ is an S-term called the *dependent function type*.

[1] For details on how to download the Coq script file for Chapters 2 and 3, see [26].

[2] For the time being, the elements on these syntactical classes should be interpreted as *symbols*.

- If v is a variable, and T, τ are S-terms, then $(\lambda \mathsf{v} : \mathsf{T}.\ \tau)$ is an S-term called a *lambda expression*.
- If τ and ν are S-terms, then $(\tau\ \nu)$ is an S-term called *function application* or just *application*.
- If v is a variable, and T, R are S-terms, then $(\sum_{\mathsf{v}:\mathsf{T}} \mathsf{R})$ is an S-term called the *dependent pair type*.
- If τ and ν are S-terms, then (τ, ν) is an S-term called a *tuple* or a *pair*.
- If v, w, y are variables, and C, τ, ν are S-terms, then $(\mathsf{ind}_{\Sigma}[\mathsf{v}.\mathsf{C}](\mathsf{w}.\mathsf{y}.\tau;\ \nu))$ is an S-term called the Σ-*eliminator*.
- If T and R are S-terms, then $(\mathsf{T} + \mathsf{R})$ is an S-term called the *sum type*.
- If τ is an S-term, then $(\mathsf{inl}\ \tau)$ and $(\mathsf{inr}\ \tau)$ are S-terms called *left injection* and *right injection*, respectively.
- If v, w, y are variables, and C, τ, ν, α are S-terms, then $(\mathsf{ind}_{+}[\mathsf{v}.\mathsf{C}](\mathsf{w}.\tau;\ \mathsf{y}.\nu;\ \alpha))$ is an S-term called the $+$-*eliminator*.
- The symbol $\mathbb{0}$ is an S-term called the *empty type*.
- If v is a variable, and C, τ are S-terms, then $(\mathsf{ind}_{\mathbb{0}}[\mathsf{v}.\mathsf{C}](\tau))$ is an S-term called the $\mathbb{0}$-*eliminator*.
- The symbol $\mathbb{1}$ is an S-term called the *unit type*.
- The symbol $\star$ is an S-term called the *unit term*.
- If v is a variable, and C, τ, ν are S-terms, then $(\mathsf{ind}_{\mathbb{1}}[\mathsf{v}.\mathsf{C}](\tau;\ \nu))$ is an S-term called the $\mathbb{1}$-*eliminator*.
- The symbol $\mathbb{N}$ is an S-term called the *natural numbers type*.
- The symbol 0 is an S-term called the *zero*.
- If τ is an S-term, then $(\mathsf{succ}\ \tau)$ is an S-term called the *successor* of τ.
- If v, w, y are variables, and C, τ, ν, α are S-terms, then $(\mathsf{ind}_{\mathbb{N}}[\mathsf{v}.\mathsf{C}](\tau;\ \mathsf{w}.\mathsf{y}.\nu;\ \alpha))$ is an S-term called the $\mathbb{N}$-*eliminator*.
- If τ, ν, and T are S-terms, then $(\tau =_{\mathsf{T}} \nu)$ is an S-term called the *identity type*.
- If T and τ are S-terms, then $(\mathsf{refl}_{\mathsf{T}}\ \tau)$ is an S-term called the *reflexivity proof* for τ.
- If v, w, y, z are variables, and C, τ, ν, α, β are S-terms, then $(\mathsf{ind}_{=}[\mathsf{v}.\mathsf{w}.\mathsf{y}.\mathsf{C}](\mathsf{z}.\tau;\ \nu;\ \alpha;\ \beta))$ is an S-term called the $=$-*eliminator*.
- If τ and ν are S-terms, then $(\mathsf{funext}\ \tau\ \nu)$ is an S-term called the *function extensionality proof*.
- If T and R are S-terms, then $(\mathsf{univalence}\ \mathsf{T}\ \mathsf{R})$ is an S-term called the *univalence proof*.
- The symbol $\mathsf{propres}$ is an S-term called the *propositional resizing proof*.

▲

For example, valid S-terms are (even if some of these S-terms are "nonsensical"):

- $(\mathbb{N}, \star)$
- $(\lambda \mathsf{v} : (\mathbb{0} + (0 =_{\mathbb{1}} \mathbb{N}))).\ (\mathsf{inl}\ \mathsf{v}))$
- $(\star =_{\mathbb{N}} \mathbb{0})$
- $(\prod_{\mathsf{v}:\mathcal{U}_1} ((\mathsf{succ}\ \mathbb{N}) =_{\mathsf{w1}} \mathsf{v}))$

As these examples show, writing an S-term involves the use of lots of parentheses. As we proceed, many conventions will be added so that we will be able to remove unnecessary parentheses. For example, the S-term $((\lambda \mathsf{n} : \mathbb{N}.\ (\mathsf{succ}\ \mathsf{n}))\ (\mathsf{succ}\ 0))$ will be written as $(\lambda \mathsf{n} : \mathbb{N}.\ \mathsf{succ}\ \mathsf{n})\ (\mathsf{succ}\ 0)$ once the appropriate conventions are introduced.

We present our first convention.

Convention 1.2.2. Given an S-term τ, we will omit the most external parentheses in τ, as long as τ is not a tuple. However, external parentheses will be used for emphasis or when there is ambiguity.

For example, $(\prod_{\mathsf{v}:\mathcal{U}_1} (\mathbb{N} =_{\mathsf{w1}} \mathsf{v}))$ can be written as $\prod_{\mathsf{v}:\mathcal{U}_1} (\mathbb{N} =_{\mathsf{w1}} \mathsf{v})$, but the tuple $(\mathbb{N}, \star)$ will be written as such. ▲

Our second syntactical class corresponds to the set of *syntactical contexts*, which we inductively define as follows.

Definition 1.2.3. The set of *syntactical contexts* $\mathcal{CNTX}$ (or *S-contexts* for short) is defined inductively by the following clauses:

- The symbol $\odot$ is an S-context called the *empty context*.
- If Γ is an S-context, v a variable, and T an S-term, then $(\Gamma,\ \mathsf{v}\colon \mathsf{T})$ is an S-context.

▲

An example of S-context is:

$$(((\odot,\ \mathsf{v1}\colon \mathbb{N}),\ \mathsf{z}\colon \mathbb{0}),\ \mathsf{v1}\colon \mathsf{z} =_{\mathbb{1}} \mathsf{v1})$$

To simplify this ugly expression, we introduce another convention.

Convention 1.2.4. In an S-context, the empty context and parentheses will be removed, so that an S-context like:

$$(\ldots(((\odot,\ \mathsf{v1}\colon \mathsf{T}_1),\ \mathsf{v2}\colon \mathsf{T}_2),\ \mathsf{v3}\colon \mathsf{T}_3),\ldots)$$

can be written as:

$$\mathsf{v1}\colon \mathsf{T}_1,\ \mathsf{v2}\colon \mathsf{T}_2,\ \mathsf{v3}\colon \mathsf{T}_3,\ldots$$

▲

So, our example above can be rewritten as:

$$\mathsf{v1}\colon \mathbb{N},\ \mathsf{z}\colon \mathbb{0},\ \mathsf{v1}\colon \mathsf{z} =_{\mathbb{1}} \mathsf{v1}$$

The two syntactical classes ($\mathcal{TERM}$ and $\mathcal{CNTX}$) come together into the concept of *judgment*. A judgment will capture the idea of "well-formedness" or "meaningfulness". HoTT uses three kinds of judgments, which we define next.

Definition 1.2.5. A *judgment* is any one of the following three kinds of sequences of symbols:

- If Γ is an S-context, then:
$$\Gamma\ \mathsf{ctx}$$
is a *context judgment*.
- If Γ is an S-context, τ is an S-term, and T is an S-term, then:
$$\Gamma \vdash \tau\colon \mathsf{T}$$
is a *typing judgment*.
- If Γ is an S-context, τ is an S-term, ν is an S-term, and T is an S-term, then:
$$\Gamma \vdash \tau \equiv \nu\colon \mathsf{T}$$
is a *definitional equality judgment*.

▲

Convention 1.2.6. For typing and definitional equality judgments, we write $\vdash \tau\colon \mathsf{T}$ and $\vdash \tau \equiv \nu\colon \mathsf{T}$, instead of $\odot \vdash \tau\colon \mathsf{T}$ and $\odot \vdash \tau \equiv \nu\colon \mathsf{T}$, respectively.

▲

Informally speaking, a context judgment $\Gamma\ \mathsf{ctx}$ will express that Γ "makes sense", it is "meaningful", or "well-formed". Intuitively, a meaningful S-context will represent a list of assumptions. For example:

$$\mathsf{w}\colon \mathsf{A},\ \mathsf{y}\colon \mathsf{B},\ \mathsf{z}\colon \mathsf{C}$$

will read as: "Let us suppose we have three terms,[3] which we name by variables w, y, and z. These terms have types[4] A, B, and C, respectively". We will explain shortly how the judgment $\Gamma\ \mathsf{ctx}$ makes precise this idea.

Similarly, a typing judgment $\Gamma \vdash \tau\colon \mathsf{T}$ will express that under the list of assumptions in Γ, term τ can be assigned type T, or that τ is well-typed, or "makes sense". Again, we will explain shortly how this idea of meaningfulness is made precise.

However, it is useful to try to think of $\Gamma \vdash \tau\colon \mathsf{T}$ as saying that term τ *is* a T, or that τ *gets classified* as a T, or that τ *belongs* to T, if we imagine type T to be some sort of collection. In fact, at this moment in the development of our type theory, it is harmless to think that *types are collections classifying terms*. Later, when our type theory is more developed, we will have to change this view of "type = collection" into a view where types have more structure than just a mere collection of terms.[5]

There is even another interpretation for $\Gamma \vdash \tau\colon \mathsf{T}$ due to the Curry-Howard correspondence [18]. A reader familiar with formal logic may have noticed the striking resemblance between typing judgments $\Gamma \vdash \tau\colon \mathsf{T}$ and sequents $\Gamma \vdash \mathsf{T}$, which are used in formal logic to state that under the assumptions in Γ, formula or proposition T has a proof. The Curry-Howard correspondence expresses that *types correspond to propositions*.

In formal logic, the notion of proof is meta-theoretical (i.e. it is a notion built outside the inference rules of the logical calculus), but type theory introduces an internal notion of proof, effectively expressing that *terms correspond to proofs of propositions*.

In this way, judgment $\Gamma \vdash \tau\colon \mathsf{T}$ now reads: "Under the list of assumptions in Γ, term τ is a proof for proposition T". This is why it is said that type theory is *proof-relevant*, because it has a built-in notion of proof, contrary to standard formal logic, where the notion of proof is external.

Finally, a definitional equality judgment $\Gamma \vdash \tau \equiv \nu\colon \mathsf{T}$ will express that under the list of assumptions in Γ, term τ can be *simplified/expanded* or *computes "by definition"* to term ν, while respecting their common type T. Hence, definitional equality judgments introduce a notion of valid or meaningful computation into the type theory.

But how can a judgment capture the idea of "meaningfulness" if there is nothing so far forbidding us from constructing a nonsensical judgment? For example, we can form the judgment $\vdash \mathbb{N}\colon \mathbb{N}$, which expresses that the collection of natural numbers *is* a natural number. The notion of *inference rule* will solve this problem.

The general form of an inference rule can be described as follows:

$$\frac{\mathcal{J}_1 \qquad \mathcal{J}_1 \qquad \mathcal{J}_2 \qquad \dots \qquad \mathcal{J}_n}{\mathcal{C}}\ \text{NAME}$$

where $\mathcal{J}_1$, $\mathcal{J}_1$, $\mathcal{J}_2$, …, $\mathcal{J}_n$, and $\mathcal{C}$ are judgments of any of the three kinds in Definition 1.2.5, and NAME is an identifier we assign to the inference rule. The judgments $\mathcal{J}_1$, $\mathcal{J}_1$, $\mathcal{J}_2$, …, $\mathcal{J}_n$ are called *premises*, and the judgment $\mathcal{C}$ is called the *conclusion*. Notice that n may be zero, corresponding to an inference rule without premises.

The premises correspond to conditions that must be satisfied in order to derive the conclusion. This means that inference rules without premises allow the derivation of $\mathcal{C}$, because there are no conditions to satisfy.

For example, let us say we want to represent natural numbers as S-terms. We know the zero symbol $\mathsf{0}$ is a natural number. Therefore, we add the rule:

$$\frac{}{\vdash \mathsf{0}\colon \mathbb{N}}\ \text{R-0}$$

so that we will be able to derive the judgment $\vdash \mathsf{0}\colon \mathbb{N}$, because the rule has no premises.

But $\mathsf{v}\colon \mathbb{N} \vdash \mathsf{0}\colon \mathbb{N}$ should also be a derivable judgment, since $\mathsf{0}$ is still a natural number independently of what list of assumptions we place in our S-context to the left of the turnstile $\vdash$ symbol. However, R-0 concludes that $\mathsf{0}$ is a natural number only when the list of assumptions is empty. In fact, $\Gamma \vdash \mathsf{0}\colon \mathbb{N}$ should be a derivable judgment

[3] We have not defined what a term is, but think of it as a "well-formed" S-term (see Definition 1.2.7).

[4] Again, we have not defined what a type is, but think of it as a "well-formed" S-term that represents a collection of terms that "look alike". In our example, $\mathsf{w}\colon \mathsf{A}$ means that w is an arbitrary term in the collection A. In this sense, types will be a kind of classification mechanism for terms. See Section 1.3 for a formal definition of type.

[5] In Section 1.7 we will see that a type is like a topological space.

for *any* S-context Γ. If we attempt to fix our rule R-$\mathsf{0}$ to include this possibility, we soon realize we need to add one inference rule for every possible S-context!

This is fixed by using a *schema*. A schema is an inference rule with placeholders. If we replace a placeholder with a specific symbol, it will produce an *instance* of the schema or an *instantiated* inference rule. In this way, a schema can encode an infinite number of inference rules in a compact way.

In our example, we change our rule R-$\mathsf{0}$ into the schema:

$$\frac{}{\Gamma \vdash \mathsf{0} : \mathbb{N}}\ \text{R-}\mathsf{0}$$

by stipulating that Γ stands for *any* S-context (we were lazy and assigned the same name R-$\mathsf{0}$ to our schema).

Following this strategy, we can add another schema expressing how the successor of a natural number can be constructed:

$$\frac{\Gamma \vdash \mathsf{n} : \mathbb{N}}{\Gamma \vdash \mathsf{succ}\ \mathsf{n} : \mathbb{N}}\ \text{R-}\mathsf{succ}$$

by stipulating that Γ stands for *any* S-context, and n stands for *any* S-term.

One possible instance of R-succ is the inference rule:

$$\frac{\vdash \mathbb{N} : \mathbb{N}}{\vdash \mathsf{succ}\ \mathbb{N} : \mathbb{N}}$$

This instantiated inference rule seems to suggest that the nonsensical judgment $\vdash \mathsf{succ}\ \mathbb{N} : \mathbb{N}$ will be derivable. But this is not the case. To derive $\vdash \mathsf{succ}\ \mathbb{N} : \mathbb{N}$, we first have to derive the premise $\vdash \mathbb{N} : \mathbb{N}$, which is impossible, since it does not fit the conclusion of neither of the rules R-succ nor R-$\mathsf{0}$ under any instance of Γ and n. In fact, it should not be hard to convince ourselves that the only judgments that can be derived by using rules R-$\mathsf{0}$ and R-succ, are judgments of the form:

$$\Gamma \vdash \mathsf{succ}\ (\mathsf{succ}\ (\ldots(\mathsf{succ}\ \mathsf{0})\ldots)) : \mathbb{N}$$

where symbol succ is applied a finite number of times (even zero times) to the symbol $\mathsf{0}$.

In other words, if R-succ and R-$\mathsf{0}$ are the only inference rules in our type theory, then the only meaningful S-terms (or the only ones that "make sense") are $(\mathsf{succ}\ (\mathsf{succ}\ (\ldots(\mathsf{succ}\ \mathsf{0})\ldots)))$. Any other combination of S-terms will be nonsensical, according to this limited type theory.

Hence, inference rules encode a notion of how we can construct *evidence* of "meaningfulness",[6] while a judgment is a *claim* of "meaningfulness". Only those judgments derivable by inference rules will have their claim confirmed. Therefore, *the only judgments we will consider to be valid are precisely the ones that can be derived by inference rules.*

We have talked informally about derivations, but derivations can be organized graphically into *derivation trees*. This concept is better explained with an example.

Let us suppose we want to derive the judgment $\vdash \mathsf{succ}\ (\mathsf{succ}\ (\mathsf{succ}\ \mathsf{0})) : \mathbb{N}$ in the type theory with rules R-$\mathsf{0}$ and R-succ above. Then, by pasting the conclusions of one instantiated inference rule after another, we have the tree (which is degenerated into one branch):

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \mathsf{0} : \mathbb{N}}\ \text{R-}\mathsf{0}}{\vdash \mathsf{succ}\ \mathsf{0} : \mathbb{N}}\ \text{R-}\mathsf{succ}}{\vdash \mathsf{succ}\ (\mathsf{succ}\ \mathsf{0}) : \mathbb{N}}\ \text{R-}\mathsf{succ}}{\vdash \mathsf{succ}\ (\mathsf{succ}\ (\mathsf{succ}\ \mathsf{0})) : \mathbb{N}}\ \text{R-}\mathsf{succ}$$

where the judgment we wanted to derive is at the *root* of the tree (i.e. the conclusion at the bottom).

Therefore, instantiated inference rules can be pasted together to produce derivation trees:

$$\dfrac{\dfrac{\dfrac{}{\mathcal{J}_1}\ \text{A-1} \qquad \dfrac{}{\mathcal{J}_2}\ \text{A-2}}{\mathcal{J}_3}\ \text{R-2} \qquad \dfrac{}{\mathcal{J}_4}\ \text{A-3}}{\mathcal{J}_5}\ \text{R-1}$$

[6] Of course, this is at the level of syntax, but we will see that at the level of semantics inference rules can encode many different concepts.

Intuitively, a derivation tree is a *proof* for the judgment at the root of the tree ($\mathcal{J}_5$ in the example).

It would be overkill to prove all theorems on this report by using derivation trees, since derivation trees tend to obfuscate the proof with irrelevant technicalities that do not add anything to its understanding. Therefore, we need a way to build judgment proofs in natural language, as it is done in standard mathematical practice. For judgments of the form $\Gamma\ \mathsf{ctx}$ this will be developed in Section 1.3. For judgments of the form $\Gamma \vdash \tau \equiv \nu : \mathsf{T}$ this will be developed in Section 1.4. For judgments of the form $\Gamma \vdash \tau : \mathsf{T}$ this will be developed starting on Section 1.5.

Before ending this section, we define some terminology we will keep using on this report.

Definition 1.2.7. We define a list of concepts involving derivation of judgments:

- We will say that a judgment $\mathcal{J}$ is *derivable* or *provable* if there is a derivation tree with $\mathcal{J}$ as its root.
- We will say that an S-context Γ is a *context* or a *meaningful S-context*, if the judgment $\Gamma\ \mathsf{ctx}$ is derivable.
- Given an S-context Γ, we will say that an S-term τ is a *term* or a *meaningful S-term*, if there is an S-term T such that the judgment $\Gamma \vdash \tau : \mathsf{T}$ is derivable.

▲

The rest of this chapter will be focused on adding inference rules, so that we can encode a notion of meaningfulness or "makes sense" for S-contexts, S-terms, and computational steps in our type theory.

1.3 Type universes and contexts

By Definition 1.2.1, for every natural number $i \geq 0$ there is a symbol $\mathcal{U}_i$ in $\mathcal{TERM}$ called a *type universe at level* i.

At this moment, it is useful to think of a type universe as a "collection of types". In other words, given an S-context Γ, if we can derive the judgment $\Gamma \vdash \tau : \mathcal{U}_i$ for some $i \geq 0$ and S-term τ, then τ *is* a type itself, since the intuitive meaning of $\Gamma \vdash \tau : \mathcal{U}_i$ is that τ belongs to $\mathcal{U}_i$. This intuition is formalized in the following definition.

Definition 1.3.1. Given an S-context Γ, we will say that an S-term T is a *type*, if there is some natural number $i \geq 0$ such that the judgment $\Gamma \vdash \mathsf{T} : \mathcal{U}_i$ is derivable.

▲

We make some important comments regarding this definition.

Remark 1.3.2. In Section 1.2, we explained that the meaning of judgment $\Gamma \vdash \tau : \mathsf{T}$ is: "τ has type T". However, Definition 1.3.1 does not allow calling T a type unless we can derive the judgment $\Gamma \vdash \mathsf{T} : \mathcal{U}_i$ for some $i \geq 0$.

This is not a problem, because HoTT has the following property:[7] "If judgment $\Gamma \vdash \tau : \mathsf{T}$ is derivable, then judgment $\Gamma \vdash \mathsf{T} : \mathcal{U}_i$ is also derivable for some $i \geq 0$". Therefore, whenever we have a derivation for the judgment $\Gamma \vdash \tau : \mathsf{T}$, we are justified in calling T a type.

The same will happen to judgments of the form $\Gamma \vdash \tau \equiv \nu : \mathsf{T}$, i.e. HoTT has the following property: "If $\Gamma \vdash \tau \equiv \nu : \mathsf{T}$ is derivable, then $\Gamma \vdash \mathsf{T} : \mathcal{U}_i$ is derivable for some $i \geq 0$".

We have a similar behavior for judgments of the form $\Gamma\ \mathsf{ctx}$, i.e. "If $\Gamma\ \mathsf{ctx}$ is derivable, then, for each assumption $\mathsf{w} : \mathsf{T}$ in Γ, we have that $\Gamma \vdash \mathsf{T} : \mathcal{U}_i$ is derivable for some $i \geq 0$". Hence, for each of the assumptions $\mathsf{w} : \mathsf{T}$ in Γ, we are justified in calling T a type.

▲

We now introduce more terminology regarding types.

Definition 1.3.3. We define a list of concepts involving types:

- Given an S-context Γ and a type T, we will say that T is *inhabited* if there is an S-term τ such that the judgment $\Gamma \vdash \tau : \mathsf{T}$ is derivable.

  Under the Curry-Howard correspondence, the statement "T is inhabited" corresponds to the statement "proposition T has a proof".

[7] Although we will not prove this property, it can be carried out by structural induction on derivation trees once all inference rules are presented.

- Given an S-context Γ and S-terms τ and T, we will say that $\tau : \mathsf{T}$ is *well-typed* if the judgment $\Gamma \vdash \tau : \mathsf{T}$ is derivable.

▲

Before introducing inference rules for type universes, we state a convention for introducing inference rules and schemas on this chapter.

Convention 1.3.4. We will introduce schemas (inference rules) by using the following pattern:

- "Schema name in natural language" ("placeholders organized into syntactical classes"):

$$\frac{\mathcal{J}_1 \quad \mathcal{J}_1 \quad \mathcal{J}_2 \quad \dots \quad \mathcal{J}_n \quad \{S_1\} \quad \dots \quad \{S_m\}}{\mathcal{C}}\ \text{IDENTIFIER}$$

After the schema name, we will indicate in parentheses the symbols acting as placeholders and to which syntactical class they belong.

Here, $\mathcal{J}_1, \dots, \mathcal{J}_n, \mathcal{C}$ are judgments (n may be zero), and $S_1, \dots, S_m$ are *side conditions* (m may be zero). A side condition will be a syntactical requirement any instance of the schema must satisfy. Side conditions will always be written inside braces { } and they will never be written in derivation trees.

▲

Using this convention, type universes are subject to the following schemas.

- Universe introduction (S-context: Γ):

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathcal{U}_i : \mathcal{U}_{i+1}}\ \mathcal{U}\text{-INTRO}$$

Informal reading. Under any context Γ, a type universe at level i *is a type* at level $i+1$.

In informal proofs we will say: "By the universe introduction rule, we know $\mathcal{U}_i : \mathcal{U}_{i+1}$", where context Γ will be left implicit, as it is usually done in standard mathematical practice.

- Cumulative universe (S-context: Γ; S-term: A):

$$\frac{\Gamma \vdash \mathsf{A} : \mathcal{U}_i}{\Gamma \vdash \mathsf{A} : \mathcal{U}_{i+1}}\ \mathcal{U}\text{-CUMUL}$$

Informal reading. If A is a type at level i, then A is also a type at level $i+1$.

In other words, a type universe includes all types belonging to universes below it, so that we can imagine type universes to be organized into some kind of containment hierarchy (if we imagine them as collections):

$$\mathcal{U}_0 \subseteq \mathcal{U}_1 \subseteq \mathcal{U}_2 \subseteq \dots \subseteq \mathcal{U}_i \subseteq \dots$$

In informal proofs we will say: "Since we know $\mathsf{A} : \mathcal{U}_i$, we will have $\mathsf{A} : \mathcal{U}_{i+1}$ by the cumulative universe rule", where context Γ is left implicit again.

This containment hierarchy is necessary, because having a type theory with a "universe of all types" $\mathcal{U}$ together with a rule stating that $\mathcal{U}$ is a type itself (i.e. an inference rule with conclusion $\mathcal{U} : \mathcal{U}$), produces an inconsistent type theory (see [15]).

Remark 1.3.5. Notice that the cumulative universe rule does not include the judgment $\Gamma\ \mathsf{ctx}$ as one of its premises. This is due to a property of HoTT, which states: "If $\Gamma \vdash \mathsf{t} : \mathsf{T}$ is derivable, then $\Gamma\ \mathsf{ctx}$ is also derivable".

In other words, we cannot derive a typing judgment without constructing a well-formed context in the first place. Hence, the premise $\Gamma\ \mathsf{ctx}$ is unnecessary in the cumulative universe rule, because the condition $\Gamma \vdash \mathsf{A} : \mathcal{U}_i$ is already stated.

A similar property holds for definitional equality judgments, i.e. if $\Gamma \vdash \mathsf{t} \equiv \mathsf{r} : \mathsf{T}$ is derivable, then $\Gamma\ \mathsf{ctx}$ is also derivable.

Therefore, from now on, the condition $\Gamma\ \mathsf{ctx}$ will be omitted as long as there is a judgment of the form $\Gamma \vdash \mathsf{t} : \mathsf{T}$ or $\Gamma \vdash \mathsf{t} \equiv \mathsf{r} : \mathsf{T}$ as one of the premises in the inference rule.

▲

Inference rules for deriving context judgments can now be stated.

- Empty context:

$$\frac{}{\odot\ \mathsf{ctx}}\ \text{CTX-EMP}$$

Informal reading. The S-context $\odot$ is always a context.

To properly state the next rule, we need to define the notion of "domain of an S-context", which can be informally described as the set of all declared variables in the S-context.

Definition 1.3.6 (Assumptions and domain of an S-context)**.** Given an S-context Γ, we define its *set of assumptions*, denoted $assum(\Gamma)$, by structural recursion on Γ as:

$$assum(\odot) = \emptyset$$
$$assum(\ (\Gamma,\ \mathsf{x}\colon \mathsf{T})\ ) = assum(\Gamma) \cup \{(\mathsf{x}, \mathsf{T})\}$$

Also, given an S-context, we define its *domain*, denoted $dom(\Gamma)$, to be the set:

$$dom(\Gamma) = \{\mathsf{w} \in \mathcal{VAR} \mid (\mathsf{w}, \mathsf{T}) \in assum(\Gamma)\}$$

i.e. $dom(\Gamma)$ is the set of all variables declared in the assumptions of Γ.

▲

We can now state the inference rule.

- Context extension (S-context: Γ; S-term: A; Variable: w):

$$\frac{\Gamma \vdash \mathsf{A}\colon \mathcal{U}_i \qquad \{\mathsf{w} \notin dom(\Gamma)\}}{\Gamma, \mathsf{w}\colon \mathsf{A}\ \mathsf{ctx}}\ \text{CTX-EXT}$$

Informal reading. If under context Γ we can construct type A, then augmenting Γ with the assumption $\mathsf{w}\colon \mathsf{A}$ forms a context again (or a valid list of assumptions), as long as w is a new variable (i.e. a variable not listed in Γ).

For example, judgment $(\mathsf{w}\colon \mathcal{U}_0,\ \mathsf{y}\colon \mathcal{U}_1)\ \mathsf{ctx}$ is derivable by the following derivation tree:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\odot\ \mathsf{ctx}}\ \text{CTX-EMP}}{\vdash \mathcal{U}_0 : \mathcal{U}_1}\ \mathcal{U}\text{-INTRO}}{(\mathsf{w}\colon \mathcal{U}_0)\ \mathsf{ctx}}\ \text{CTX-EXT}}{\mathsf{w}\colon \mathcal{U}_0 \vdash \mathcal{U}_1 : \mathcal{U}_2}\ \mathcal{U}\text{-INTRO}}{(\mathsf{w}\colon \mathcal{U}_0,\ \mathsf{y}\colon \mathcal{U}_1)\ \mathsf{ctx}}\ \text{CTX-EXT}$$

Instead of using derivation trees to prove context judgments, we will use informal reasoning as follows. Let us suppose we want to prove the context judgment:

$$(\mathsf{x}_1\colon \mathsf{T}_1,\ \mathsf{x}_2\colon \mathsf{T}_2,\ \ldots,\ \mathsf{x}_{n-1}\colon \mathsf{T}_{n-1},\ \mathsf{x}_n\colon \mathsf{T}_n)\ \mathsf{ctx}$$

We ask: Can we prove that T_1 is a type from no assumptions at all? If the answer is yes, then the assumption $\mathsf{x}_1\colon \mathsf{T}_1$ gets justified (by implicit application of the CTX-EXT rule).

Now, we ask: Can we prove that T_2 is a type under the assumption $\mathsf{x}_1\colon \mathsf{T}_1$? If the answer is yes, then the assumption $\mathsf{x}_2\colon \mathsf{T}_2$ gets justified (of course, we need to verify that variable x_2 is different from x_1).

Hence, the general step is: Can we prove that T_j is a type under the assumptions $\mathsf{x}_1\colon \mathsf{T}_1$, ..., $\mathsf{x}_{j-1}\colon \mathsf{T}_{j-1}$? If the answer is yes, then the assumption $\mathsf{x}_j\colon \mathsf{T}_j$ gets justified, as long as variable x_j is different from each one of $\mathsf{x}_1, \ldots, \mathsf{x}_{j-1}$.

This process stops until $\mathsf{x}_n\colon \mathsf{T}_n$ gets justified.

By using this informal procedure, we can prove that judgment $(\mathsf{w}\colon \mathcal{U}_0,\ \mathsf{y}\colon \mathcal{U}_1)\ \mathsf{ctx}$ is derivable as follows.

Under no assumptions, by the universe introduction rule, we know $\mathcal{U}_0 : \mathcal{U}_1$. Therefore, we can make the assumption $\mathsf{w} : \mathcal{U}_0$. Now, knowing assumption $\mathsf{w} : \mathcal{U}_0$, by the universe introduction rule we know $\mathcal{U}_1 : \mathcal{U}_2$.[8] Therefore, we can make the assumption $\mathsf{y} : \mathcal{U}_1$. This proves that judgment $(\mathsf{w} : \mathcal{U}_0,\ \mathsf{y} : \mathcal{U}_1)\ \mathsf{ctx}$ is derivable.

Checking that an S-context is well-formed is so straightforward that most of the time we will omit the proof, unless there is some assumption $\mathsf{x_j} : \mathsf{T_j}$ for which it is not obvious that $\mathsf{T_j}$ is a type. On those rare cases, we will have to provide a proof that $\mathsf{T_j}$ is a type.

1.4 Structural rules

Let us suppose we want to prove that judgment $(\mathsf{A} : \mathcal{U}_0,\ \mathsf{w} : \mathsf{A})\ \mathsf{ctx}$ is derivable (here, A is a variable).

If we proceed informally, we say: "Under no assumptions, by the universe introduction rule we know $\mathcal{U}_0 : \mathcal{U}_1$. Therefore, we can make the assumption $\mathsf{A} : \mathcal{U}_0$. Now, given assumption $\mathsf{A} : \mathcal{U}_0$, we can conclude $\mathsf{A} : \mathcal{U}_0$ since it is an assumption we already had (in other words, we concluded that A is a type). Therefore, we can add the assumption $\mathsf{w} : \mathsf{A}$. This proves that $(\mathsf{A} : \mathcal{U}_0,\ \mathsf{w} : \mathsf{A})\ \mathsf{ctx}$ is derivable."

However, this informal "proof" is wrong. The problem is at step: "Now, given assumption $\mathsf{A} : \mathcal{U}_0$, we can conclude $\mathsf{A} : \mathcal{U}_0$ since it is an assumption we already had". There is no inference rule allowing the conclusion of an assumption!

In fact, if we attempt to build a derivation tree for $(\mathsf{A} : \mathcal{U}_0,\ \mathsf{w} : \mathsf{A})\ \mathsf{ctx}$, we will get stuck:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\odot\ \mathsf{ctx}}\ \text{CTX-EMP}}{\vdash \mathcal{U}_0 : \mathcal{U}_1}\ \mathcal{U}\text{-INTRO}}{(\mathsf{A} : \mathcal{U}_0)\ \mathsf{ctx}}\ \text{CTX-EXT}}{\mathsf{A} : \mathcal{U}_0 \vdash \mathsf{A} : \mathcal{U}_0}\ (????)}{(\mathsf{A} : \mathcal{U}_0,\ \mathsf{w} : \mathsf{A})\ \mathsf{ctx}}\ \text{CTX-EXT}$$

where (????) is a nonexistent rule allowing the conclusion $\mathsf{A} : \mathcal{U}_0 \vdash \mathsf{A} : \mathcal{U}_0$ from $(\mathsf{A} : \mathcal{U}_0)\ \mathsf{ctx}$.

We will call this desired rule the "variable rule" (in logic, this rule is usually called the "assumption rule").

- Variable (S-context: Γ; S-term: T; Variable: w):

$$\dfrac{\Gamma\ \mathsf{ctx} \qquad \{(\mathsf{w}, \mathsf{T}) \in assum(\Gamma)\}}{\Gamma \vdash \mathsf{w} : \mathsf{T}}\ \text{VBLE}$$

  Informal reading. Under any context Γ, we can always conclude an assumption $\mathsf{w} : \mathsf{T}$ that is already in context Γ.

With this rule, the informal proof we did above for $(\mathsf{A} : \mathcal{U}_0,\ \mathsf{w} : \mathsf{A})\ \mathsf{ctx}$ becomes a correct proof. This example also shows that the variable rule is never explicitly used in informal proofs.

The next rule states that known facts remain true even when we augment our assumptions.

- Weakening (S-context: Γ, Δ; S-term: τ, T):

$$\dfrac{\Delta\ \mathsf{ctx} \qquad \Gamma \vdash \tau : \mathsf{T} \qquad \{assum(\Gamma) \subseteq assum(\Delta)\}}{\Delta \vdash \tau : \mathsf{T}}\ \text{WEAK}$$

  Informal reading. If under some context Γ we are able to prove that τ has type T, then this fact will still hold if we add more assumptions to context Γ.

In informal arguments, the weakening rule is used when proving subproofs or claims inside other proofs. For example, let us suppose we are in the middle of a proof (under some assumptions Γ) where we were able to prove $\tau : \mathsf{T}$ as some intermediate step. Now, let us suppose we want to prove a claim inside our proof. Normally, we

[8] Notice that $\mathcal{U}_1 : \mathcal{U}_2$ is true even without the assumption $\mathsf{w} : \mathcal{U}_0$. We say that assumption $\mathsf{w} : \mathcal{U}_0$ is irrelevant for concluding that $\mathcal{U}_1$ is a type.

would start the proof of the claim by stating some hypotheses. These extra hypotheses will implicitly add more assumptions into Γ, since the assumptions we made for the "external" proof are still in place. However, while proving the claim, we want to be able to use the result $\tau : \mathsf{T}$ which was proved before starting the claim. The weakening rule will allow the use of $\tau : \mathsf{T}$ inside the proof of our claim, since $\tau : \mathsf{T}$ will still hold under the extended list of assumptions.

Hence, the weakening rule will never be explicitly stated in informal proofs, as it captures the intuitive idea that proved statements remain true even if we start a subproof or a claim.

Now, it is time to add rules involving definitional equality judgments (i.e. judgments of the form $\Gamma \vdash \tau \equiv \nu : \mathsf{T}$). These rules will capture our intuition that $\equiv$ (definitional equality) behaves like identity in standard mathematics, i.e. if $\Gamma \vdash \tau \equiv \nu : \mathsf{T}$ is derivable, then we should be able to *freely replace* τ anywhere for term ν, and vice versa. Also, $\equiv$ will capture the idea that τ *simplifies/expands by definition* into ν, or that τ and ν are *interchangeable*.

The first three rules express that definitional equality is an equivalence relation on terms.

- Reflexivity of $\equiv$ (S-context: Γ; S-term: τ, T):

$$\frac{\Gamma \vdash \tau : \mathsf{T}}{\Gamma \vdash \tau \equiv \tau : \mathsf{T}} \text{ REFL}$$

  Informal reading. If we can construct a term τ, then τ simplifies/expands to τ.

- Symmetry of $\equiv$ (S-context: Γ; S-term: τ, ν, T):

$$\frac{\Gamma \vdash \tau \equiv \nu : \mathsf{T}}{\Gamma \vdash \nu \equiv \tau : \mathsf{T}} \text{ SYM}$$

  Informal reading. If term τ simplifies/expands to ν, then ν simplifies/expands to τ.

- Transitivity of $\equiv$ (S-context: Γ; S-term: τ, ν, ρ, T):

$$\frac{\Gamma \vdash \tau \equiv \nu : \mathsf{T} \qquad \Gamma \vdash \nu \equiv \rho : \mathsf{T}}{\Gamma \vdash \tau \equiv \rho : \mathsf{T}} \text{ TRAN}$$

  Informal reading. If term τ is interchangeable with ν and term ν is interchangeable with ρ, then τ is interchangeable with ρ.

The next two rules express that this equivalence relation is respected by typing.

- Type interchangeability 1 (S-context: Γ; S-term: τ, T, R):

$$\frac{\Gamma \vdash \mathsf{T} \equiv \mathsf{R} : \mathcal{U}_i \qquad \Gamma \vdash \tau : \mathsf{T}}{\Gamma \vdash \tau : \mathsf{R}} \text{ T-EQUIV-1}$$

  Informal reading. If types T and R are interchangeable, then any term τ with type T can also be considered to have type R.

- Type interchangeability 2 (S-context: Γ; S-term: τ, ν, T, R):

$$\frac{\Gamma \vdash \mathsf{T} \equiv \mathsf{R} : \mathcal{U}_i \qquad \Gamma \vdash \tau \equiv \nu : \mathsf{T}}{\Gamma \vdash \tau \equiv \nu : \mathsf{R}} \text{ T-EQUIV-2}$$

  Informal reading. If types T and R are interchangeable, then any two interchangeable terms τ and ν of type T can also be considered to have type R.

We also have weakening for definitional equality judgments, which captures the same idea as weakening for typing judgments (i.e. the WEAK rule).

- Weakening for $\equiv$ (S-context: Γ, Δ; S-term: τ, ν, T):

$$\frac{\Delta\ \mathsf{ctx} \qquad \Gamma \vdash \tau \equiv \nu : \mathsf{T} \qquad \{assum(\Gamma) \subseteq assum(\Delta)\}}{\Delta \vdash \tau \equiv \nu : \mathsf{T}}\ \text{WEAK-EQUIV}$$

Informal reading. If under some context Γ we are able to prove that τ is interchangeable with ν, then this fact will still hold if we add more assumptions to context Γ.

In informal proofs, all the previous rules for definitional equality are used without explicit mention. For example, suppose we have the following derivation tree, where $(*)$ indicates omitted parts of the derivation:

$$\dfrac{\dfrac{(*)}{\Gamma \vdash \mathsf{t}_1 \equiv \mathsf{t}_2 : \mathsf{T}} \qquad \dfrac{\dfrac{(*)}{\Gamma \vdash \mathsf{t}_2 \equiv \mathsf{t}_3 : \mathsf{T}} \qquad \dfrac{(*)}{\Gamma \vdash \mathsf{t}_3 \equiv \mathsf{t}_4 : \mathsf{T}}\ \text{TRAN}}{\Gamma \vdash \mathsf{t}_2 \equiv \mathsf{t}_4 : \mathsf{T}}}{\Gamma \vdash \mathsf{t}_1 \equiv \mathsf{t}_4 : \mathsf{T}}\ \text{TRAN}$$

In an informal proof, this derivation tree is translated to a sequence of equalities:

$$\mathsf{t}_1 \equiv \mathsf{t}_2 \equiv \mathsf{t}_3 \equiv \mathsf{t}_4 \tag{1.1}$$

without explicit mention to the transitivity rule. The justification of each equality is usually omitted unless it is not obvious. A similar remark applies to the reflexivity and symmetry rules.

Also, notice that neither the context nor the type of terms t_1, t_2, t_3, t_4 in (1.1) were written, as these two pieces of data are left implicit in informal proofs.

However, the reader should have in mind that whenever we write equations like (1.1), we are really claiming that judgment $\Gamma \vdash \mathsf{t}_1 \equiv \mathsf{t}_4 : \mathsf{T}$ is derivable.

As an example of the informal use of the type interchangeability rules, suppose that we were able to prove $\tau : \mathsf{T}$. Suppose we also proved the following sequence of equalities:

$$\mathsf{T} \equiv \mathsf{T}_2 \equiv \ldots \equiv \mathsf{R}$$

Then, we are allowed to say: "We have $\tau : \mathsf{T}$, or equivalently, $\tau : \mathsf{R}$". This is just an application of the T-EQUIV-1 rule, because T and R are interchangeable types. The use of the type interchangeability 2 rule is similar.

There are more rules involving definitional equality judgments, but we defer them to Section 1.5 because we require to introduce types into our theory to properly state them.

Before ending this section, we introduce conventions regarding the use of definitions in informal proofs. By a *definition* we mean a symbol that serves as a shortcut for a longer term, so that certain expressions become easier to read.

Convention 1.4.1. Definitions inside informal proofs will be introduced by the pattern:

$$\mathsf{Identifier} :\equiv \tau$$

where $\mathsf{Identifier}$ is the name given to the definition, and τ is the S-term acting as the definition body.

When we want to emphasize that τ is actually a term, we will write:

$$\mathsf{Identifier} :\equiv \tau : \mathsf{T}$$

where T must be a type according to the current context, which is always left implicit.

▲

Bear in mind that definitions are *not* new symbols added into the theory. In other words, defined symbols occurring inside a term should be thought as abbreviations.

For example, if we define:

$$\mathsf{univ} :\equiv \mathcal{U}_0 : \mathcal{U}_1$$

Then, the S-term:

$$\mathcal{U}_2 + \mathsf{univ}$$

really stands for:

$$\mathcal{U}_2 + \mathcal{U}_0$$

Therefore, if we claim in an informal proof that the following holds by definition:

$$\mathsf{univ} \equiv \mathcal{U}_0$$

What we are really claiming is that the following holds:

$$\mathcal{U}_0 \equiv \mathcal{U}_0$$

which is simply an application of the reflexivity rule!

1.5 Types

It is time to add types into our theory. We need inference rules expressing how to form a type, how to build its terms, how terms on the type can be decomposed to be able to construct proofs, and how terms on the type can simplify. As such, the inference rules will be grouped into rule classes. Each rule class represents an action that can be done on the type:

- *Formation rules*: They will state the valid way to form the type in question.
- *Introduction rules*: They will express how terms in the type can be built. These rules are also called the type *constructors*.
- *Elimination rules*: They will express how to use elements of the type to construct functions and proofs. These rules are also called the type *eliminators*.
- *Computation rules*: They will express what happens when a constructor is given to an eliminator. They capture a notion of term simplification.
- *Uniqueness rules*: These rules will be optional. They will express what happens when an eliminator is given to a constructor.

In addition to these rules, there will be *congruence rules*, which express that type formation, type constructors, and type eliminators respect definitional equality. Congruence rules are not placed inside a rule class because they are more like structural rules. Congruence rules express that all the rule classes above "fit together nicely" with definitional equality.

1.5.1 Dependent function type (Π-type)

A Π-type, also called a *dependent function type*, will capture the idea of *dependent functions*. A dependent function is a function whose codomain varies according to the element given as input. To motivate the concept, let us work in standard mathematics.

Suppose we have the indexed family of sets $\{\mathbb{N}^i\}_{i\in\mathbb{N}}$, where each $\mathbb{N}^i$ is the i-times Cartesian product of the natural numbers $\mathbb{N}$, i.e. an element of $\mathbb{N}^i$ is a tuple of natural numbers with exactly i elements.

Suppose we define a function f that takes a natural number n and returns the tuple $(n, n, \ldots, n) \in \mathbb{N}^n$ where n is repeated n times. In standard notation, we would write the domain and codomain of this function as $f : \mathbb{N} \to \bigcup_{i\in\mathbb{N}} \mathbb{N}^i$ with the following side note: "for every $n \in \mathbb{N}$, we have $f(n) \in \mathbb{N}^n$".

In other words, given 1 as input, function f outputs a 1-tuple: $f(1) \in \mathbb{N}^1$; given 2 as input, f outputs a 2-tuple: $f(2) \in \mathbb{N}^2$; and so on. Hence, the "codomain" of f depends on the input: for 1, the "codomain" is $\mathbb{N}^1$; for 2, the "codomain" is $\mathbb{N}^2$; etc. We say that f is a dependent function.

The standard notation for describing the domain and (varying) codomain of function f is a bit clumsy, because we have to say that the codomain is $\bigcup_{i\in\mathbb{N}} \mathbb{N}^i$ and then add the side note: "for every $n \in \mathbb{N}$, we have $f(n) \in \mathbb{N}^n$".

Let us suppose we change the notation from $f : \mathbb{N} \to \bigcup_{i\in\mathbb{N}} \mathbb{N}^i$ to $f : \prod_{n\in\mathbb{N}} \mathbb{N}^n$ so that this new notation *emphasizes* the fact that the codomain changes to $\mathbb{N}^n$ on input $n \in \mathbb{N}$. This is the idea behind a Π-type.

In fact, the notation $\prod_{n\in\mathbb{N}} \mathbb{N}^n$ was chosen to suggest the Cartesian product of the indexed family $\{\mathbb{N}^i\}_{i\in\mathbb{N}}$, as defined in set theory:

$$\prod_{n\in\mathbb{N}} \mathbb{N}^n = \left\{ g : \mathbb{N} \to \bigcup_{i\in\mathbb{N}} \mathbb{N}^i \;\middle|\; \forall n \in \mathbb{N}.\ g(n) \in \mathbb{N}^n \right\}$$

which happily corresponds to the set of dependent functions from $\mathbb{N}$ to the family $\{\mathbb{N}^i\}_{i\in\mathbb{N}}$. This is why a Π-type is also called a *dependent product type*, because it corresponds to a Cartesian product of an indexed family when types are interpreted as sets.

Notice that dependent functions can also represent standard functions (i.e. functions with a fixed codomain). For example, given sets A and B, a function $f : A \to B$ can be represented as $f : \prod_{i\in A} B$ where the same codomain B is assigned to each input $i \in A$.

We now present inference rules for working with Π-types.

- Formation (S-context: Γ; S-term: A, B; Variable: w):

$$\frac{\Gamma \vdash \mathsf{A} : \mathcal{U}_i \qquad \Gamma, \mathsf{w} : \mathsf{A} \vdash \mathsf{B} : \mathcal{U}_i}{\Gamma \vdash \prod_{\mathsf{w}:\mathsf{A}} \mathsf{B} : \mathcal{U}_i}\ \Pi\text{-FORM}$$

  Informal reading. Let A be a type at level i. If given $\mathsf{w} : \mathsf{A}$ we are able to construct a type B at level i, then $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ is a type at level i.

  Notice that variable w may appear in type B. This means that type B will change if variable w is replaced by a specific term. So, the expression $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ captures the idea that the codomain will change if the input is changed.

In standard mathematics, we usually use the notation $x \mapsto \tau$ to define a function that takes x as input and returns an expression τ. For example, the successor function $S : \mathbb{N} \to \mathbb{N}$ is defined by $x \mapsto x + 1$.

In HoTT, to define a function we use a *lambda expression*. A lambda expression, written $(\lambda \mathsf{w} : \mathsf{T}.\ \tau)$, will denote a function that takes as input a w of type T, and returns a term τ.

The following introduction rule will make precise how lambda expressions represent dependent functions.

- Introduction (S-context: Γ; S-term: τ, A, B; Variable: w):

$$\frac{\Gamma, \mathsf{w} : \mathsf{A} \vdash \tau : \mathsf{B}}{\Gamma \vdash (\lambda \mathsf{w} : \mathsf{A}.\ \tau) : \prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}}\ \Pi\text{-INTRO}$$

  Informal reading. If given $\mathsf{w} : \mathsf{A}$ we are able to construct a term τ of type B (where τ and B are dependent on variable w), then we can construct the lambda expression $(\lambda \mathsf{w} : \mathsf{A}.\ \tau)$, which denotes a dependent function transforming terms of type A into terms of type B.

  Again, notice that variable w may appear in term τ and type B. The fact that variable w appears in type B means that the codomain of function $(\lambda \mathsf{w} : \mathsf{A}.\ \tau)$ will vary depending on the specific term given as input.

  The introduction rule states that type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ has dependent functions as elements.

To define non-dependent functions (i.e. functions with fixed codomain), we need to introduce the concepts of *bind structure*, and *bound* and *free* occurrence of a variable in an S-term.

Definition 1.5.1 (Occurrence of a variable)**.** An *occurrence* of a variable w in an S-term is an appearance of w in the S-term. Occurrences are counted from left to right.

For example, variable w has two occurrences in $(\lambda \mathsf{w} : \mathsf{A}.\ \mathbb{1} + \mathsf{w})$, where the first occurrence is at $(\lambda \mathsf{w} \ldots$, and the second occurrence is at $\ldots + \mathsf{w})$.

▲

Definition 1.5.2 (Bind structures and local variables for Π-types)**.** There are two kinds of *bind structures* for Π-types.

- In a lambda expression $(\lambda \mathsf{w}\colon \mathsf{A}.\ \mathsf{t})$, a *bind structure* is defined by the underlined parts, as indicated in the following expression:

  $$\underline{\lambda \mathsf{w}\colon}\ \mathsf{A}.\ \underline{\mathsf{t}}$$

  The set of *local variables* in the bind structure is defined to be $\{\mathsf{w}\}$.

  For example, in the lambda expression $(\lambda \mathsf{w}\colon \mathsf{y}.\ \mathsf{w} =_{\mathsf{y}} \mathsf{w})$, the first occurrence of y is *not* in the bind structure, but the second occurrence of y *is* in the bind structure.

- In a Π-type expression $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$, a *bind structure* is defined by the underlined parts, as indicated in the following expression:

  $$\underline{\prod}_{\underline{\mathsf{w}\colon}\,\mathsf{A}}\,\underline{\mathsf{B}}$$

  The set of *local variables* in the bind structure is defined to be $\{\mathsf{w}\}$.

▲

Remark 1.5.3. As more types are introduced, more bind structures with their corresponding set of local variables will be added.

Also, if S is a bind structure, we will denote its set of local variables by $LV(\mathsf{S})$.

▲

Definition 1.5.4 (Free and bound occurrence of a variable)**.** Let w be a variable and τ an S-term.

- We will say that the n-th occurrence of w in τ is *bound* if there is some bind structure S in τ such that the n-th occurrence of w is inside S and $\mathsf{w} \in LV(\mathsf{S})$.
- We will say that the n-th occurrence of w in τ is *free* if it is not bound, or equivalently, whenever S is a bind structure in τ such that the n-th occurrence of w is inside S, we have that $\mathsf{w} \notin LV(\mathsf{S})$.
- Variable w *occurs free* in τ if there is an occurrence of w in τ which is free.
- Variable w *does not occur free* in τ if every occurrence of w in τ is not free, or equivalently, if every occurrence of w in τ is bound.

  Notice that if variable w does not occur at all in τ, then w does not occur free in τ, since the condition holds vacuously.

▲

For example, let us analyze the (nonsensical) S-term $(\lambda \mathsf{w}\colon \mathsf{w}.\ \mathsf{z}\ (\lambda \mathsf{y}\colon \mathsf{w}.\ \mathsf{y} + \mathsf{z}))$, where $\mathsf{w}, \mathsf{y}, \mathsf{z}$ are variables. We will denote by (1) the bind structure defined by the most external lambda, and by (2) the bind structure defined by the most internal lambda.

- The first occurrence of w is bound (i.e. not free), because it occurs inside bind structure (1), and w is a local variable in (1).
- The second occurrence of w is free because the condition holds vacuously, as the second occurrence of w is not inside bind structures (1) and (2).
- The third occurrence of w is bound, because it occurs inside bind structure (1), and w is a local variable in (1).

  Notice that this occurrence of w is not inside bind structure (2).
- The first occurrence of y is bound, because it occurs inside bind structure (2), and y is a local variable in (2).

  Notice that this occurrence of y is also inside bind structure (1), but y is not a local variable in (1).

- The second occurrence of y is bound, because it occurs inside bind structure (2), and y is a local variable in (2).

  Notice that this occurrence of y is also inside bind structure (1), but y is not a local variable in (1).
- The first occurrence of z is free. Although it occurs inside bind structure (1), z is not a local variable in (1).
- The second occurrence of z is free. Although it occurs inside bind structures (1) and (2), z is not a local variable in neither of them.
- Variable w occurs free, since its second occurrence is free.
- Variable y does not occur free, since all its occurrences are bound.
- Variable z occurs free, since its first (and second) occurrence is free.
- Any other variable $\mathsf{t} \in \mathcal{VAR}$ does not occur free, since t does not occur at all in the expression (i.e. the condition holds vacuously).

With this at hand, we now introduce the *non-dependent function type* ($\to$), also called *function type* or *arrow type*, as special case of the dependent function type.

Definition 1.5.5 (Arrow type $\to$)**.** Let A and B be S-terms. Let w be a variable. If variable w does *not occur free* in B, the S-term $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ will be written as $\mathsf{A} \to \mathsf{B}$.

The S-term $\mathsf{A} \to \mathsf{B}$ should be interpreted as the type of all functions with domain A and codomain B.

▲

As an example on how to define functions, let us define the identity function.

Definition 1.5.6 (Identity function)**.** Let $\mathsf{A} : \mathcal{U}_i$ be a type. Then, we define the function:[9]

$$\mathsf{id}_{\mathsf{A}} :\equiv (\lambda \mathsf{w} : \mathsf{A}.\ \mathsf{w}) : \mathsf{A} \to \mathsf{A}$$

which will be called the *identity function* over A.[10]

▲

So, the identity function just returns what it receives as input. How can we be sure that this "definition" written in natural language is justified by a judgment derivation in our type theory? Observe that the definition starts by introducing the assumption $\mathsf{A} : \mathcal{U}_i$. Hence, it is implicitly building a context. Then, it goes to claim that term $(\lambda \mathsf{w} : \mathsf{A}.\ \mathsf{w})$ has type $\mathsf{A} \to \mathsf{A}$.

In other words, our "definition" is actually a claim stating that judgment $\mathsf{A} : \mathcal{U}_i \vdash (\lambda \mathsf{w} : \mathsf{A}.\ \mathsf{w}) : \mathsf{A} \to \mathsf{A}$ is derivable, or equivalently, that judgment $\mathsf{A} : \mathcal{U}_i \vdash (\lambda \mathsf{w} : \mathsf{A}.\ \mathsf{w}) : \prod_{\mathsf{w}:\mathsf{A}} \mathsf{A}$ is derivable. Therefore, we need to check the derivability of this judgment if we want to be sure that our definition "makes sense".

By using derivation trees, the proof looks like:

$$\dfrac{\dfrac{\dfrac{(*)}{(\mathsf{A} : \mathcal{U}_i,\ \mathsf{w} : \mathsf{A})\ \mathsf{ctx}}\ \text{CTX-EXT}}{\mathsf{A} : \mathcal{U}_i,\ \mathsf{w} : \mathsf{A} \vdash \mathsf{w} : \mathsf{A}}\ \text{VBLE}}{\mathsf{A} : \mathcal{U}_i \vdash (\lambda \mathsf{w} : \mathsf{A}.\ \mathsf{w}) : \prod\limits_{\mathsf{w}:\mathsf{A}} \mathsf{A}}\ \Pi\text{-INTRO} \tag{1.2}$$

where $(*)$ is a proof for the judgment $(\mathsf{A} : \mathcal{U}_i,\ \mathsf{w} : \mathsf{A})\ \mathsf{ctx}$, which we omit.

However, an informal proof reads: "First, assume $\mathsf{A} : \mathcal{U}_i$, which is clearly a context. Now, by the Π-INTRO rule, it is enough to assume $\mathsf{w} : \mathsf{A}$ and prove $\mathsf{w} : \mathsf{A}$ (checking of course that w is different from A to get a valid context). But $\mathsf{w} : \mathsf{A}$ trivially holds because it is now one of our assumptions!"

[9] Notice we are justified in writing $\mathsf{A} \to \mathsf{A}$ instead of $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{A}$ because variable w does not occur in variable A, as they are different variables.

[10] Strictly speaking, we are introducing a different symbol id_{A} for each type $\mathsf{A} : \mathcal{U}_i$.

Most of the time we will not check that definitions are correct (i.e. we will omit the proof of the judgment they claim), as a quick glance will suffice to convince us that the involved terms are well-typed. Of course, there will be cases where this is not obvious and we will have to justify the definition with an argument starting like: "Observe this definition is well-typed because...".

There is another observation regarding the identity function. Notice that the derivation tree (1.2) ended with the judgment:

$$\mathsf{A}:\mathcal{U}_i \vdash (\lambda \mathsf{w}:\mathsf{A}.\ \mathsf{w}):\prod_{\mathsf{w}:\mathsf{A}} \mathsf{A}$$

but we can apply the Π-INTRO rule again to get the judgment:

$$\vdash (\lambda \mathsf{A}:\mathcal{U}_i.\ (\lambda \mathsf{w}:\mathsf{A}.\ \mathsf{w})):\prod_{\mathsf{A}:\mathcal{U}_i}\left(\prod_{\mathsf{w}:\mathsf{A}} \mathsf{A}\right)$$

or equivalently:

$$\vdash (\lambda \mathsf{A}:\mathcal{U}_i.\ (\lambda \mathsf{w}:\mathsf{A}.\ \mathsf{w})):\prod_{\mathsf{A}:\mathcal{U}_i} (\mathsf{A}\to\mathsf{A}) \tag{1.3}$$

which represents a function that takes a type A as input, and *returns the identity function* on A. Hence, functions can produce other functions as output and, as we will see shortly, can receive other functions as input.

Judgment (1.3) also serves to introduce more conventions.

Convention 1.5.7 (Parentheses)**.** We follow these conventions regarding parentheses:

- In a lambda expression $\lambda \mathsf{w}:\mathsf{T}.\ \tau$, we will omit the most external parentheses in τ and T.
- In a dependent function type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ we will omit the most external parentheses in A and B.

  There is, however, an exception to this rule. In an S-term of the form $\mathsf{A}\to\mathsf{B}$ we will *never* remove the most external parentheses in A and B. But, if B has the form $\mathsf{C}\to\mathsf{D}$, then the most external parentheses in B can be removed.

  For example, the S-term $\mathsf{A}\to(\mathsf{C}\to\mathsf{D})$ will be written as $\mathsf{A}\to\mathsf{C}\to\mathsf{D}$. Instead, the parentheses in $(\mathsf{A}\to\mathsf{C})\to\mathsf{D}$ cannot be removed.

Of course, we may add external parentheses for emphasis or when there is ambiguity.

▲

Hence, judgment (1.3) can we written as:

$$\vdash (\lambda \mathsf{A}:\mathcal{U}_i.\ \lambda \mathsf{w}:\mathsf{A}.\ \mathsf{w}):\prod_{\mathsf{A}:\mathcal{U}_i} \mathsf{A}\to\mathsf{A}$$

Another convention involves the use of nested lambdas.

Convention 1.5.8 (Nested lambdas)**.** If we have an S-term of the form:

$$\lambda \mathsf{a}:\mathsf{A}.\ \lambda \mathsf{b}:\mathsf{B}.\ \lambda \mathsf{c}:\mathsf{C}.\ \ldots \lambda \mathsf{q}:\mathsf{Q}.\ \tau$$

which corresponds to nested lambdas, the S-term can be written as:

$$\lambda(\mathsf{a}:\mathsf{A})(\mathsf{b}:\mathsf{B})(\mathsf{c}:\mathsf{C})\ldots(\mathsf{q}:\mathsf{Q}).\ \tau$$

▲

Hence, judgment (1.3) can be written as:

$$\vdash (\lambda(\mathsf{A}:\mathcal{U}_i)(\mathsf{w}:\mathsf{A}).\ \mathsf{w}):\prod_{\mathsf{A}:\mathcal{U}_i} \mathsf{A}\to\mathsf{A}$$

Another convention involves variables in dependent function types $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$.

Convention 1.5.9 (Grouping of variables)**.** If we have an S-term of the form:

$$\prod_{a_1:A}\prod_{a_2:A}\cdots\prod_{a_n:A}\prod_{b_1:B}\prod_{b_2:B}\cdots\prod_{b_m:B}\prod_{w:A}\prod_{c:C} D$$

we may group variables as long as they have a common type and are *adjacent* to one another:

$$\prod_{a_1,a_2,\ldots,a_n:A}\prod_{b_1,b_2,\ldots,b_m:B}\prod_{w:A}\prod_{c:C} D$$

Notice we did not place variable w at the end of list $a_1, a_2, \ldots, a_n$ because variable w is not adjacent to a_n.

▲

We also have a convention regarding the definition of functions (as in the definition of the identity function).

Convention 1.5.10 (Implicit lambda notation)**.** When introducing a definition of the form:

$$\mathsf{Identifier} :\equiv \lambda(a:A)(b:B)(c:C)\ldots(q:Q).\ \tau$$

This may be written as:

$$\mathsf{Identifier}\ a\ b\ c\ \ldots\ q :\equiv \tau$$

where all input types are left implicit.

▲

Hence, the identity function can be defined as:

$$\mathsf{id}_A\ w :\equiv w$$

Now, before introducing the elimination rule for Π-types, we need to explain the concept of *capture-free substitution*.

Suppose we have an S-term τ such that variable w occurs in τ. Let us suppose we want to substitute all occurrences of w with another S-term ν. Denote by $\tau[w/\nu]$ the result of substituting in τ all occurrences of w by the S-term ν. Then, for example, $((w + \mathbb{1}) + w)[w/\mathbb{0}]$ will denote the S-term $(\mathbb{0} + \mathbb{1}) + \mathbb{0}$.

However, if we want the result of the substitution to be an S-term, then *this definition of substitution is wrong*. For example, $(\lambda A{:}\mathcal{U}_0.\ A)[A/\mathbb{0}]$ denotes $(\lambda\mathbb{0}{:}\mathcal{U}_0.\ \mathbb{0})$ which is not even an S-term because $\mathbb{0}$ is not a variable. In the lambda expression $(\lambda A{:}\mathcal{U}_0.\ A)$, variable A should not be replaceable with a non-variable S-term because A denotes an *arbitrary input* to a function. Notice that all occurrences of A are bound on this lambda expression. So, it seems that we should only replace free occurrences of variables.

We now reinterpret $\tau[w/\nu]$ to denote the result of substituting in τ all *free* occurrences of w by the S-term ν. With this, $(\lambda A{:}\mathcal{U}_0.\ A)[A/\mathbb{0}]$ correctly denotes the S-term $(\lambda A{:}\mathcal{U}_0.\ A)$, because every occurrence of A is not free.

However, even this definition has a problem. Consider lambda $(\lambda A{:}\mathcal{U}_0.\ B)$ where A and B are variables. This lambda denotes a function that receives a type and returns variable B, where variable B is declared in some context Γ that is left implicit. Now, consider the substitution $(\lambda A{:}\mathcal{U}_0.\ B)[B/A]$ which produces lambda $(\lambda A{:}\mathcal{U}_0.\ A)$, because the only occurrence of B is free. Hence, we changed our function into the identity function in $\mathcal{U}_0$. This is clearly not what we want. The meaning of our functions should not change after performing a substitution.

The problem in this last example is that the free occurrence of variable B was replaced by A, which happens to be a local variable in $(\lambda A{:}\mathcal{U}_0.\ B)$. In other words, after performing the substitution of B for A, variable A got *captured* by the local variable in $(\lambda A{:}\mathcal{U}_0.\ B)$.

This problem is solved if we rename all occurrences of local variables in the bind structure *before* performing the substitution. Therefore, to perform:

$$(\lambda A{:}\mathcal{U}_0.\ B)[B/A]$$

first, inside the lambda expression, rename all occurrences of the local variable A into a different variable:

$$(\lambda C{:}\mathcal{U}_0.\ B)[B/A]$$

and then perform the substitution of all free occurrences of B, to get:

$$\lambda \mathsf{C}{:}\mathcal{U}_0.\ \mathsf{A}$$

Why is it valid to rename local variables? Because local variables have meaning only inside the bind structure, as they behave like "placeholders". For example, the function $(\lambda \mathsf{w}{:}\mathsf{A}.\ \mathsf{w})$ will still be the identity function on A if we change variable w to variable t.

This last form of substitution (with local variable renaming) is called *capture-free substitution*, and it is the one we will use.

Definition 1.5.11 (Capture-free substitution)**.** We will denote by:

$$\tau[\mathsf{w}_1/\nu_1, \mathsf{w}_2/\nu_2, \ldots, \mathsf{w}_\mathsf{m}/\nu_\mathsf{m}]$$

the result of *simultaneously*[11] substituting in τ all *free* occurrences of w_1, w_2, …, w_m by the S-terms ν_1, ν_2, …, ν_m, respectively, with the added side note that all local variables on all bind structures inside τ should be renamed (only if necessary) into variables not appearing in S-terms ν_1, ν_2, …, ν_m.

Also, an expression like:

$$\tau[\mathsf{w}_1/\nu_1][\mathsf{w}_2/\nu_2][\ldots][\mathsf{w}_\mathsf{m}/\nu_\mathsf{m}] \tag{1.4}$$

will denote:

$$(\ (\ (\tau[\mathsf{w}_1/\nu_1])[\mathsf{w}_2/\nu_2]\ )[\ldots]\ )[\mathsf{w}_\mathsf{m}/\nu_\mathsf{m}]$$

which corresponds to iterated application of substitution, i.e. we first perform $\tau[\mathsf{w}_1/\nu_1]$, then substitution w_2/ν_2 is performed on the result of the first substitution, and so on. Hence, an expression like (1.4) will denote *sequential* substitution.

▲

For example, $(\mathsf{a},\mathsf{b})[\mathsf{a}/\mathsf{b},\mathsf{b}/\mathsf{c}]$ denotes (b,c) because a and b must be substituted at the same time. Instead, $(\mathsf{a},\mathsf{b})[\mathsf{a}/\mathsf{b}][\mathsf{b}/\mathsf{c}]$ denotes the iterated substitution $((\mathsf{a},\mathsf{b})[\mathsf{a}/\mathsf{b}])[\mathsf{b}/\mathsf{c}]$. If we perform the most internal substitution, we get $(\mathsf{b},\mathsf{b})[\mathsf{b}/\mathsf{c}]$, which then produces (c,c).

We can now state the elimination rule for Π-types.

- Elimination (S-context: Γ; S-term: f, τ, A, B; Variable: w):

$$\frac{\Gamma \vdash \mathsf{f}{:}\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B} \qquad \Gamma \vdash \tau{:}\mathsf{A}}{\Gamma \vdash \mathsf{f}\ \tau{:}\mathsf{B}[\mathsf{w}/\tau]}\ \Pi\text{-ELIM}$$

  Informal reading. If we are given a dependent function $\mathsf{f}{:}\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ and a term τ of type A, then we can give τ as input to f (i.e. the application $\mathsf{f}\ \tau$) to obtain a term belonging to type $\mathsf{B}[\mathsf{w}/\tau]$.

  The elimination rule expresses that the codomain must change according to the function input. This is why we have to substitute all free occurrences of variable w in B to obtain the specific codomain corresponding to input τ.

  Also, notice that function application is written as $\mathsf{f}\ \mathsf{n}$, while in standard mathematics it is written as $f(n)$.

  When the elimination rule is instantiated with the special case of the arrow type (where variable w does not occur free in B), we get:

$$\frac{\Gamma \vdash \mathsf{f}{:}\mathsf{A} \to \mathsf{B} \qquad \Gamma \vdash \tau{:}\mathsf{A}}{\Gamma \vdash \mathsf{f}\ \tau{:}\mathsf{B}}$$

  because substitution will leave B intact, as w does not occur free in B.

[11] By "simultaneous" we mean that all variables should be replaced at the same time.

Convention 1.5.12 (Parentheses)**.** A term of the form:

$$((\ldots(((\nu_1\ \nu_2)\ \nu_3)\ \nu_4)\ldots)\ \nu_{\mathsf{m}})$$

representing a sequence of applications (i.e. ν_1 is applied on ν_2, the result of which is a function applied on ν_3, the result of which is a function applied on ν_4, and so on), will be written as:

$$\nu_1\ \nu_2\ \nu_3\ \nu_4\ \ldots\ \nu_{\mathsf{m}}$$

to emphasize the sequential nature of applications.

▲

As an example of function application, we define the function composition operator.

Definition 1.5.13 (Function composition)**.** Let $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$, and $\mathsf{C}:\mathcal{U}_k$ be types. We define *function composition* as:[12]

$$\circ :\equiv (\lambda(\mathsf{f}:\mathsf{A}\to\mathsf{B})(\mathsf{g}:\mathsf{B}\to\mathsf{C}).\ \lambda\mathsf{w}:\mathsf{A}.\ \mathsf{g}\ (\mathsf{f}\ \mathsf{w})):(\mathsf{A}\to\mathsf{B})\to(\mathsf{B}\to\mathsf{C})\to\mathsf{A}\to\mathsf{C}$$

Instead of $(\circ\ \mathsf{f}\ \mathsf{g})$, we will write $(\mathsf{g}\circ\mathsf{f})$, which uses *infix notation*. Notice that the order of functions is switched in the infix notation, to conform with standard mathematical notation.

▲

In other words, function composition receives functions $\mathsf{f}:\mathsf{A}\to\mathsf{B}$ and $\mathsf{g}:\mathsf{B}\to\mathsf{C}$, and returns as output the function $(\lambda\mathsf{w}:\mathsf{A}.\ \mathsf{g}\ (\mathsf{f}\ \mathsf{w})):\mathsf{A}\to\mathsf{C}$, which represents the composition of f followed by g.

We can write a small informal proof to verify that $\circ$ is well-typed: "Suppose $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$, and $\mathsf{C}:\mathcal{U}_k$. By the Π-INTRO rule applied three times, it is enough to suppose $\mathsf{f}:\mathsf{A}\to\mathsf{B}$, $\mathsf{g}:\mathsf{B}\to\mathsf{C}$, and $\mathsf{w}:\mathsf{A}$, and then prove $(\mathsf{g}\ (\mathsf{f}\ \mathsf{w})):\mathsf{C}$. But, by two applications of the Π-ELIM rule, we have $(\mathsf{f}\ \mathsf{w}):\mathsf{B}$, and $(\mathsf{g}\ (\mathsf{f}\ \mathsf{w})):\mathsf{C}$".

However, there is an important observation regarding our informal proof. To justify that the following is a context:

$$\mathsf{A}:\mathcal{U}_i,\ \mathsf{B}:\mathcal{U}_j,\ \mathsf{C}:\mathcal{U}_k,\ \mathsf{f}:\mathsf{A}\to\mathsf{B},\ \mathsf{g}:\mathsf{B}\to\mathsf{C},\ \mathsf{w}:\mathsf{A}$$

we have to prove that $\mathsf{A}\to\mathsf{B}$ and $\mathsf{B}\to\mathsf{C}$ are types, each at *some* universe level.

Let us check the case for $\mathsf{A}\to\mathsf{B}$, as the case for other type is similar. We have $\mathsf{A}:\mathcal{U}_i$ and $\mathsf{B}:\mathcal{U}_j$ as hypotheses. Remember that $\mathsf{A}\to\mathsf{B}$ really stands for $\prod_{\mathsf{z}:\mathsf{A}}\mathsf{B}$ where z is a "dummy" variable, i.e. a variable that does not occur free in B. Therefore, by the Π-FORM rule, to conclude that $\prod_{\mathsf{z}:\mathsf{A}}\mathsf{B}$ is a type, we have to prove the following:

- A is a type at some universe level m.
- Assuming $\mathsf{z}:\mathsf{A}$, prove that B is a type at the *same* universe level m.

Thus, we have to prove that A and B can be placed at the *same* universe level, but we seem to be stuck since the only thing we know is that A is a type at level i and B is a type at level j, where i and j may be different levels.

The problem is fixed by the cumulative universe rule ($\mathcal{U}$-CUMUL). By the $\mathcal{U}$-CUMUL rule we can conclude $\mathsf{A}:\mathcal{U}_{\max\{i,j\}}$ and $\mathsf{B}:\mathcal{U}_{\max\{i,j\}}$ by repeatedly applying the rule (even zero times) on A and B. Therefore, A and B can be placed at level $\max\{i,j\}$. Hence, the Π-FORM rule can conclude that $\mathsf{A}\to\mathsf{B}$ is a type at level $\max\{i,j\}$.

The next rule states how function application simplifies.

- Computation (S-context: Γ; S-term: τ, ν, A, B; Variable: w):

$$\frac{\Gamma,\mathsf{w}:\mathsf{A}\vdash\tau:\mathsf{B}\qquad\Gamma\vdash\nu:\mathsf{A}}{\Gamma\vdash(\lambda\mathsf{w}:\mathsf{A}.\ \tau)\ \nu\equiv\tau[\mathsf{w}/\nu]:\mathsf{B}[\mathsf{w}/\nu]}\ \Pi\text{-COMP}$$

Informal reading. The result of giving input ν to function $(\lambda\mathsf{w}:\mathsf{A}.\ \tau)$ is term $\tau[\mathsf{w}/\nu]$. Or equivalently, the application $(\lambda\mathsf{w}:\mathsf{A}.\ \tau)\ \nu$ simplifies to $\tau[\mathsf{w}/\nu]$.

[12] Remember that this definition is really making the claim that the following judgment is derivable:

$$\mathsf{A}:\mathcal{U}_i,\ \mathsf{B}:\mathcal{U}_j,\ \mathsf{C}:\mathcal{U}_k\vdash(\lambda(\mathsf{f}:\mathsf{A}\to\mathsf{B})(\mathsf{g}:\mathsf{B}\to\mathsf{C}).\ \lambda\mathsf{w}:\mathsf{A}.\ \mathsf{g}\ (\mathsf{f}\ \mathsf{w})):(\mathsf{A}\to\mathsf{B})\to(\mathsf{B}\to\mathsf{C})\to\mathsf{A}\to\mathsf{C}$$

In the beginning of Section 1.5 it was stated that a computation rule describes what happens when a constructor is given to an eliminator. In this case, a lambda expression (i.e. the constructor, because lambda expressions are produced by an introduction rule) is placed inside a function application (i.e. the eliminator, because applications are produced by an elimination rule).

For example, with the computation rule, the identity function behaves as expected. Given $\mathsf{A} : \mathcal{U}_i$ and $\mathsf{a} : \mathsf{A}$, we have by definition and the Π-COMP rule:[13]

$$\mathsf{id}_{\mathsf{A}}\ \mathsf{a} \equiv (\lambda \mathsf{w}.\ \mathsf{w})\ \mathsf{a} \equiv \mathsf{w}[\mathsf{w}/\mathsf{a}] \equiv \mathsf{a} \tag{1.5}$$

Notice we wrote $(\lambda \mathsf{w}.\ \mathsf{w})$ instead of $(\lambda \mathsf{w} : \mathsf{A}.\ \mathsf{w})$ in (1.5). We will usually remove type information from lambdas so that the expressions become shorter.

Also, the composition operator computes as expected. Given $\mathsf{A} : \mathcal{U}_i$, $\mathsf{B} : \mathcal{U}_j$, $\mathsf{C} : \mathcal{U}_k$, $\mathsf{h} : \mathsf{A} \to \mathsf{B}$, and $\mathsf{k} : \mathsf{B} \to \mathsf{C}$, we have by definition and the Π-COMP rule:

$$\begin{aligned}
\mathsf{k} \circ \mathsf{h} \equiv \circ\ \mathsf{h}\ \mathsf{k} &\equiv (\lambda \mathsf{f}.\ \lambda \mathsf{g}.\ \lambda \mathsf{w}.\ \mathsf{g}\ (\mathsf{f}\ \mathsf{w}))\ \mathsf{h}\ \mathsf{k} \\
&\equiv (\lambda \mathsf{g}.\ \lambda \mathsf{w}.\ \mathsf{g}\ (\mathsf{f}\ \mathsf{w}))[\mathsf{f}/\mathsf{h}]\ \mathsf{k} \\
&\equiv (\lambda \mathsf{g}.\ \lambda \mathsf{w}.\ \mathsf{g}\ (\mathsf{h}\ \mathsf{w}))\ \mathsf{k} \\
&\equiv (\lambda \mathsf{w}.\ \mathsf{g}\ (\mathsf{h}\ \mathsf{w}))[\mathsf{g}/\mathsf{k}] \\
&\equiv \lambda \mathsf{w}.\ \mathsf{k}\ (\mathsf{h}\ \mathsf{w})
\end{aligned}$$

And so, given $\mathsf{a} : \mathsf{A}$, we have:

$$(\mathsf{h} \circ \mathsf{k})\ \mathsf{a} \equiv (\lambda \mathsf{w}.\ \mathsf{h}\ (\mathsf{k}\ \mathsf{w}))\ \mathsf{a} \equiv (\mathsf{h}\ (\mathsf{k}\ \mathsf{w}))[\mathsf{w}/\mathsf{a}] \equiv \mathsf{h}\ (\mathsf{k}\ \mathsf{a})$$

From now on, we will omit any step where a substitution is involved, unless otherwise stated for emphasis. For example, instead of:

$$\mathsf{id}_{\mathsf{A}}\ \mathsf{a} \equiv (\lambda \mathsf{w}.\ \mathsf{w})\ \mathsf{a} \equiv \mathsf{w}[\mathsf{w}/\mathsf{a}] \equiv \mathsf{a}$$

we will write:

$$\mathsf{id}_{\mathsf{A}}\ \mathsf{a} \equiv (\lambda \mathsf{w}.\ \mathsf{w})\ \mathsf{a} \equiv \mathsf{a}$$

The next rule states that a function is determined by its values.

- Uniqueness (S-context: Γ; S-term: f, A, B; Variable: w):

$$\frac{\Gamma \vdash \mathsf{f} : \prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}}{\Gamma \vdash (\lambda \mathsf{w} : \mathsf{A}.\ \mathsf{f}\ \mathsf{w}) \equiv \mathsf{f} : \prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}}\ \Pi\text{-UNIQ}$$

Informal reading. Given a dependent function f, "wrapping" function f inside a lambda expression does not produce a function different from f.

In the beginning of Section 1.5 it was stated that a uniqueness rule describes what happens when an eliminator is given to a constructor. In this case, a function application (i.e. the eliminator) is the body of a lambda expression (i.e. the constructor).

The next rules are the so-called "congruence rules" for Π-types. They express that Π-type formation, lambda formation and function application respect definitional equality. More intuitively, these rules express that interchangeable terms are formed from interchangeable subterms. This idea is so intuitive that congruence rules are never explicitly used in informal proofs. Also, for other types, the pattern these rules follow will be the same. Therefore, *congruence rules will not be explicitly stated for the remaining types on this chapter.*

[13] These equations are an informal proof for the following judgment:

$$\mathsf{A} : \mathcal{U}_i,\ \ \mathsf{a} : \mathsf{A} \vdash \mathsf{id}_{\mathsf{A}}\ \mathsf{a} \equiv \mathsf{a} : \mathsf{A}$$

- Formation congruence (S-context: Γ; S-term: A_1, A_2, B_1, B_2; Variable: w):

$$\frac{\Gamma \vdash \mathsf{A}_1 \equiv \mathsf{A}_2 : \mathcal{U}_i \qquad \Gamma, \mathsf{w}\colon \mathsf{A}_1 \vdash \mathsf{B}_1 \equiv \mathsf{B}_2 : \mathcal{U}_i}{\Gamma \vdash \prod_{\mathsf{w}:\mathsf{A}_1} \mathsf{B}_1 \equiv \prod_{\mathsf{w}:\mathsf{A}_2} \mathsf{B}_2 : \mathcal{U}_i}\ \Pi\text{-FORM-EQUIV}$$

  Informal reading. Two Π-types are interchangeable if their domains and codomains are interchangeable.

- Introduction congruence (S-context: Γ; S-term: τ_1, τ_2, A_1, A_2, B_1, B_2; Variable: w):

$$\frac{\Gamma \vdash \mathsf{A}_1 \equiv \mathsf{A}_2 : \mathcal{U}_i \qquad \Gamma, \mathsf{w}\colon \mathsf{A}_1 \vdash \mathsf{B}_1 \equiv \mathsf{B}_2 : \mathcal{U}_i \qquad \Gamma, \mathsf{w}\colon \mathsf{A}_1 \vdash \tau_1 \equiv \tau_2 : \mathsf{B}_1}{\Gamma \vdash \lambda \mathsf{w}\colon \mathsf{A}_1.\ \tau_1 \equiv \lambda \mathsf{w}\colon \mathsf{A}_2.\ \tau_2 : \prod_{\mathsf{w}:\mathsf{A}_1} \mathsf{B}_1}\ \Pi\text{-INTRO-EQUIV}$$

  Informal reading. Two lambdas are interchangeable if the types of their inputs are interchangeable and their bodies are interchangeable.

- Elimination congruence (S-context: Γ; S-term: f, g, τ_1, τ_2, A, B; Variable: w):

$$\frac{\Gamma \vdash \mathsf{f} \equiv \mathsf{g} : \prod_{\mathsf{w}:\mathsf{A}} \mathsf{B} \qquad \Gamma \vdash \tau_1 \equiv \tau_2 : \mathsf{A}}{\Gamma \vdash \mathsf{f}\ \tau_1 \equiv \mathsf{g}\ \tau_2 : \mathsf{B}[\mathsf{w}/\tau_1]}\ \Pi\text{-ELIM-EQUIV}$$

  Informal reading. Two function applications are interchangeable if both functions are interchangeable and both inputs are interchangeable.

In the beginning of this section, we showed an example of a dependent function with domain and codomain described as $f : \prod_{n\in\mathbb{N}} \mathbb{N}^n$. Observe that this notation shows the family index n in the codomain of function f. But in the notation for type theory, type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ does not make explicit that B depends on variable w. It would be nice to have a notation emphasizing the dependence of type B on variable w. For example, something like $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}\ \mathsf{w}$, where $\mathsf{B}\ \mathsf{w}$ is an application. For this, we need the concept of type family.

Definition 1.5.14 (Type family)**.** Let $\mathsf{A} : \mathcal{U}_i$ be a type. Any function of type $\mathsf{A} \to \mathcal{U}_j$ will be called a *type family* indexed by A.

We will say that a type family $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$ is *constant*, if P is a constant function, i.e. P has the form $\lambda \mathsf{z}\colon \mathsf{A}.\ \mathsf{D}$ for some $\mathsf{D} : \mathcal{U}_j$, such that variable z does not occur free in D.

▲

With this, the formation, introduction, and elimination rules can be stated in terms of type families. Please note that we are not replacing the previous inference rules. We are just adding a result on top those rules so that it is easier to reason with statements involving Π-types.

Lemma 1.5.15 (Rules with type families)**.** *Let $\mathsf{A} : \mathcal{U}_i$ be a type and $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$ a type family. Then, the formation, introduction, and elimination rules can be stated in terms of type families as follows:*

- *Formation:*

  Term $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ is a type at level $\max\{i, j\}$.

- *Introduction:*

  If for every $\mathsf{w}\colon \mathsf{A}$ we can construct a term $\mathsf{t}\colon \mathsf{P}\ \mathsf{w}$. Then, $(\lambda \mathsf{w}\colon \mathsf{A}.\ \mathsf{t})$ is a function of type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$.

- *Elimination:*

  If we have $\mathsf{f}\colon \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ and $\mathsf{a}\colon \mathsf{A}$, then $\mathsf{f}\ \mathsf{a}\colon \mathsf{P}\ \mathsf{a}$.

Proof. We prove each item in turn.

- Formation:

  The statement is claiming that the following judgment is derivable:

  $$\mathsf{A}:\mathcal{U}_i,\ \mathsf{P}:\mathsf{A}\to\mathcal{U}_j \vdash \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}:\mathcal{U}_{\max\{i,j\}}$$

  So, we suppose the context. To be able to use the Π-FORM rule we have to prove that:

  - Term A is a type at level $\max\{i,j\}$. Proof: by assumption we have $\mathsf{A}:\mathcal{U}_i$. Hence, by the cumulative universe rule we have $\mathsf{A}:\mathcal{U}_{\max\{i,j\}}$.
  - If we suppose $\mathsf{w}:\mathsf{A}$, then term $\mathsf{P}\ \mathsf{w}$ is a type at level $\max\{i,j\}$. Proof: by the Π-ELIM rule (for the special case of arrow types) we have $\mathsf{P}\ \mathsf{w}:\mathcal{U}_j$, and then, by the cumulative universe rule we have $\mathsf{P}\ \mathsf{w}:\mathcal{U}_{\max\{i,j\}}$.

- Introduction:

  The statement is claiming that the following inference rule holds:

  $$\frac{\mathsf{A}:\mathcal{U}_i,\ \mathsf{P}:\mathsf{A}\to\mathcal{U}_j,\ \mathsf{w}:\mathsf{A}\vdash \mathsf{t}:\mathsf{P}\ \mathsf{w}}{\mathsf{A}:\mathcal{U}_i,\ \mathsf{P}:\mathsf{A}\to\mathcal{U}_j \vdash (\lambda \mathsf{w}:\mathsf{A}.\ \mathsf{t}):\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}}$$

  But it is simply an instance of the Π-INTRO rule.

- Elimination:

  The statement is claiming that the following judgment is derivable:

  $$\mathsf{A}:\mathcal{U}_i,\ \mathsf{P}:\mathsf{A}\to\mathcal{U}_j,\ \mathsf{f}:\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w},\ \mathsf{a}:\mathsf{A}\vdash \mathsf{f}\ \mathsf{a}:\mathsf{P}\ \mathsf{a}$$

  So, we suppose the context. But then, by Π-ELIM we have $\mathsf{f}\ \mathsf{a}:(\mathsf{P}\ \mathsf{w})[\mathsf{w}/\mathsf{a}]$. But $(\mathsf{P}\ \mathsf{w})[\mathsf{w}/\mathsf{a}]$ denotes term $\mathsf{P}\ \mathsf{a}$.

□

The following remark explains how to compute the universe level of a Π-type. This property will be used a lot on this report.

Remark 1.5.16 (Universe level for Π-types)**.** Suppose we are given a type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$. The only way to form this type is to use the Π-FORM rule, which states that types A and B must be at the *same* universe level.

Hence, if A is at level i, and for every $\mathsf{w}:\mathsf{A}$, type B is at level j, then the only way for type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ to be valid is that the cumulative universe rule was used to move types A and B up into the universe $\mathcal{U}_{\max\{i,j\}}$. In other words, we must have $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}:\mathcal{U}_{\max\{i,j\}}$.

Therefore, to compute the universe level of $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$, just take the maximum of the levels of A and B for every $\mathsf{w}:\mathsf{A}$.

▲

To end this section, we state a small remark regarding the interpretation of Π-types under the Curry-Howard correspondence.

Remark 1.5.17 (Logical interpretation of Π-types)**.** Under the Curry-Howard correspondence, a Π-type corresponds to a *universally quantified statement* [18]. In other words, type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ can be interpreted as: "For every proof w for A, proposition B holds". A function $\mathsf{f}:\prod_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ can also be seen as a procedure that transforms proofs for A into proofs for B. Therefore, function f acts as a proof for the universal statement.

The special case of the arrow type $\mathsf{A}\to\mathsf{B}$ corresponds to *logical implication*. In other words, type $\mathsf{A}\to\mathsf{B}$ can be interpreted as: "If A holds, then B holds". Again, a function $\mathsf{f}:\mathsf{A}\to\mathsf{B}$ can be a seen as a procedure that transforms proofs for A into proofs for B, making f itself a proof for the implication.

In fact, a reader familiar with formal logic may have noticed that the Π-INTRO rule corresponds to the universal generalization rule (the special case of the arrow type corresponds to the implication introduction rule); and the Π-ELIM rule corresponds to the universal specification rule (the special case of the arrow type corresponds to the Modus Ponens rule).

▲

1.5.2 Dependent pair type (Σ-type)

A Σ-type, also called a *dependent pair type*, will capture the idea of *dependent pairs*. A dependent pair is a pair (a, b) where the type of b depends on the first coordinate a.

To motivate the concept, let us work in standard mathematics. Suppose we have an indexed family of sets $\{A_i\}_{i \in I}$. The disjoint union of this family is the set:

$$\sum_{i \in I} A_i = \bigcup_{i \in I} (\{i\} \times A_i)$$

The elements of this set are pairs (a, b) where $a \in I$ and $b \in A_a$. In other words, pair (a, b) is a dependent pair because the set to which b belongs depends on the first coordinate a. Hence, $\sum_{i \in I} A_i$ is a set of dependent pairs.

If we interpret types as collections, a Σ-type will represent the disjoint union of an indexed family of collections. This is why Σ-types are also called *dependent sum types*.

When the indexed family $\{A_i\}_{i \in I}$ assigns the same set B to each $i \in I$, we get:

$$\sum_{i \in I} B = \bigcup_{i \in I} (\{i\} \times B) = I \times B$$

Hence, the binary Cartesian product will be a special case of the disjoint union of an indexed family.

The idea behind a Σ-type like $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ is that A will be the type for the first coordinate of the pair, and for each $\mathsf{w} : \mathsf{A}$, type B will be the varying type for the second coordinate of the pair.

We now present inference rules for working with Σ-types.

- Formation (S-context: Γ; S-term: A, B; Variable: w):

$$\frac{\Gamma \vdash \mathsf{A} : \mathcal{U}_i \qquad \Gamma, \mathsf{w} : \mathsf{A} \vdash \mathsf{B} : \mathcal{U}_i}{\Gamma \vdash \sum_{\mathsf{w}:\mathsf{A}} \mathsf{B} : \mathcal{U}_i} \ \Sigma\text{-FORM}$$

  Informal reading. Let A be a type at level i. If given $\mathsf{w} : \mathsf{A}$ we are able to construct a type B at level i, then $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ is a type at level i.

  Notice that variable w may appear in type B. This means that type B will change whenever variable w is replaced by a specific term. So, at least at this intuitive level, the expression $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ captures the idea that the type of the second coordinate will change if the first coordinate changes.

We introduce the *non-dependent pair type* ($\times$), also called *Cartesian product type* or just *product type*, as special case of the dependent pair type.

Definition 1.5.18 (Product type $\times$)**.** Let A and B be S-terms. Let w be a variable. If variable w does *not occur free* in B, the S-term $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ will be written as $\mathsf{A} \times \mathsf{B}$.

The S-term $\mathsf{A} \times \mathsf{B}$ should be interpreted as the type of all pairs with first coordinate in A and second coordinate in B.

▲

As in standard mathematics, to denote a pair we will use the expression (a, b), as evidenced by the following introduction rule.

- Introduction (S-context: Γ; S-term: a, b, A, B; Variable: w):

$$\frac{\Gamma, \mathsf{w} : \mathsf{A} \vdash \mathsf{B} : \mathcal{U}_i \qquad \Gamma \vdash \mathsf{a} : \mathsf{A} \qquad \Gamma \vdash \mathsf{b} : \mathsf{B}[\mathsf{w}/\mathsf{a}]}{\Gamma \vdash (\mathsf{a}, \mathsf{b}) : \sum_{\mathsf{w}:\mathsf{A}} \mathsf{B}} \ \Sigma\text{-INTRO}$$

  Informal reading. If we can construct a term a of type A and a term b whose type depends on term a, then (a, b) is a dependent pair.

The introduction rule is stating that type $\sum_{w:A} B$ has dependent pairs as elements.

When the introduction rule is instantiated with the product type (where variable w does not occur free in B), we get:

$$\frac{\Gamma, w : A \vdash B : \mathcal{U}_i \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B}{\Gamma \vdash (a, b) : A \times B}$$

because substitution will leave B intact.

We follow the next convention regarding parentheses.

Convention 1.5.19 (Parentheses)**.** In a dependent pair type $\sum_{w:A} B$ we will omit the most external parentheses in A and B.

So, an S-term like:

$$\sum_{A:(\mathcal{U}_0+\mathbb{1})} \left(\sum_{w:A} w \right)$$

can be written as:

$$\sum_{A:\mathcal{U}_0+\mathbb{1}} \sum_{w:A} w$$

There is, however, an exception to this rule. In an S-term of the form $A \times B$ we will *never* remove the most external parentheses in A and B. However, if A or B are products themselves, then their most external parentheses can be removed on the corresponding case.

For example, the S-term $A \times (C \times D)$ will be written as $A \times C \times D$, but parentheses will be kept in $(C \to D) \times B$.

Notice that there is ambiguity in the expression $A \times C \times D$, because it could be interpreted as $(A \times C) \times D$ or $A \times (C \times D)$. In Corollary 1.6.24, we will prove that the product type is associative. Therefore, it does not matter how S-term $A \times C \times D$ is interpreted. Nevertheless, for the sake of resolving ambiguity, $(A \times C) \times D$ will be the preferred interpretation.

Of course, we may add parentheses for emphasis or when there is ambiguity.

▲

Another convention involves the grouping of variables in dependent pair types.

Convention 1.5.20 (Grouping of variables)**.** If we have an S-term of the form:

$$\sum_{a_1:A} \sum_{a_2:A} \cdots \sum_{a_n:A} \sum_{b_1:B} \sum_{b_2:B} \cdots \sum_{b_m:B} \sum_{w:A} \sum_{c:C} D$$

we may group variables as long as they have a common type and are *adjacent* to one another:

$$\sum_{a_1,a_2,\ldots,a_n:A} \sum_{b_1,b_2,\ldots,b_m:B} \sum_{w:A} \sum_{c:C} D$$

Notice we did not place variable w at the end of list $a_1, a_2, \ldots, a_n$ because variable w is not adjacent to a_n.

▲

As we did for Π-types, the formation and introduction rules can be stated in terms of type families. For the moment, we are leaving out the elimination rule, as this rule will be encapsulated into an statement called "induction principle".

Lemma 1.5.21 (Rules with type families)**.** *Let $A : \mathcal{U}_i$ be a type and $P : A \to \mathcal{U}_j$ a type family. Then, the formation and introduction rules can be stated in terms of type families as follows:*

- *Formation:*

  Term $\sum_{w:A} P\, w$ is a type at level $\max\{i, j\}$.

- *Introduction:*

  If we have $a : A$ and $b : P\, a$, then $(a, b) : \sum_{w:A} P\, w$.

Proof. We prove each item in turn.

- Formation:

  The statement is claiming that the following judgment is derivable:

  $$\mathsf{A}:\mathcal{U}_i,\ \mathsf{P}:\mathsf{A}\to\mathcal{U}_j \vdash \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}:\mathcal{U}_{\max\{i,j\}}$$

  So, we suppose the context. To be able to use the Σ-FORM rule we have to prove:

  - Term A is a type at level $\max\{i,j\}$. Proof: by assumption we have $\mathsf{A}:\mathcal{U}_i$. Hence, by the cumulative universe rule we have $\mathsf{A}:\mathcal{U}_{\max\{i,j\}}$.
  - If we suppose $\mathsf{w}:\mathsf{A}$, then term $\mathsf{P}\ \mathsf{w}$ is a type at level $\max\{i,j\}$. Proof: by the Π-ELIM rule we have $\mathsf{P}\ \mathsf{w}:\mathcal{U}_j$, and then, by the cumulative universe rule we have $\mathsf{P}\ \mathsf{w}:\mathcal{U}_{\max\{i,j\}}$.

- Introduction:

  The statement is claiming that the following judgment is derivable:

  $$\mathsf{A}:\mathcal{U}_i,\ \mathsf{P}:\mathsf{A}\to\mathcal{U}_j,\ \mathsf{a}:\mathsf{A},\ \mathsf{b}:\mathsf{P}\ \mathsf{a} \vdash (\mathsf{a},\mathsf{b}):\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$$

  So, we suppose the context. To be able to use the Σ-INTRO rule, we have to prove:

  - If we suppose $\mathsf{w}:\mathsf{A}$, then $\mathsf{P}\ \mathsf{w}$ is a type at some level. Proof: just apply the Π-ELIM rule to get $\mathsf{P}\ \mathsf{w}:\mathcal{U}_j$.
  - $\mathsf{a}:\mathsf{A}$ must hold. Proof: by assumption.
  - $\mathsf{b}:(\mathsf{P}\ \mathsf{w})[\mathsf{w}/\mathsf{a}]$ must hold. Proof: term $(\mathsf{P}\ \mathsf{w})[\mathsf{w}/\mathsf{a}]$ denotes $\mathsf{P}\ \mathsf{a}$, but we have by assumption $\mathsf{b}:\mathsf{P}\ \mathsf{a}$.

□

Now, we present the elimination rule for Σ-types.

- Elimination (S-context: Γ; S-term: g, p, A, B, C; Variable: w, y, z):

  $$\frac{\Gamma,\mathsf{z}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{B}\vdash\mathsf{C}:\mathcal{U}_i \qquad \Gamma,\mathsf{w}:\mathsf{A},\mathsf{y}:\mathsf{B}\vdash\mathsf{g}:\mathsf{C}[\mathsf{z}/(\mathsf{w},\mathsf{y})] \qquad \Gamma\vdash\mathsf{p}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{B}}{\Gamma\vdash\mathsf{ind}_\Sigma[\mathsf{z}.\mathsf{C}](\mathsf{w}.\mathsf{y}.\mathsf{g};\mathsf{p}):\mathsf{C}[\mathsf{z}/\mathsf{p}]}\ \Sigma\text{-ELIM}$$

  Informal reading. This is one case where interpreting C as a proposition or predicate (allowed by the Curry-Howard correspondence) is more illuminating.

  Condition $\Gamma,\mathsf{z}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{B}\vdash\mathsf{C}:\mathcal{U}_i$ reads: "Let C be a unary predicate on dependent pairs".

  Condition $\Gamma\vdash\mathsf{p}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{B}$ reads: "Let p be an arbitrary dependent pair".

  Then, the rest reads: "To prove that predicate C holds for dependent pair p (i.e. type $\mathsf{C}[\mathsf{z}/\mathsf{p}]$ in the conclusion), prove that predicate C holds for any dependent pair of the form (w,y) (i.e. condition $\Gamma,\mathsf{w}:\mathsf{A},\mathsf{y}:\mathsf{B}\vdash\mathsf{g}:\mathsf{C}[\mathsf{z}/(\mathsf{w},\mathsf{y})]$)".

  The intuition is that if C holds for all *canonical* dependent pairs (a canonical dependent pair is a term of the form (w,y)), then it will certainly hold for an arbitrary dependent pair p, because p can be "decomposed" into a pair (r,t), for some r and t, as this is the only way to build pairs in the first place, i.e. the only way to build p is by using the introduction rule.

  The Σ-eliminator term $\mathsf{ind}_\Sigma[\mathsf{z}.\mathsf{C}](\mathsf{w}.\mathsf{y}.\mathsf{g};\mathsf{p})$ is a *witness* for this proof by elimination. It remembers all the pieces: the predicate C, the proof g that C holds for all canonical pairs, and the arbitrary pair p.

  The dot notation $\mathsf{z}.\mathsf{C}$ and $\mathsf{w}.\mathsf{y}.\mathsf{g}$ inside term $\mathsf{ind}_\Sigma[\mathsf{z}.\mathsf{C}](\mathsf{w}.\mathsf{y}.\mathsf{g};\mathsf{p})$ denotes bind structures (see Definition 1.5.23 below). The dot notation emphasizes that z is a local variable for C, and w,y are local variables for g.

Convention 1.5.22 (Parentheses)**.** Given an S-term of the form:

$$\mathsf{ind}_{\Sigma}[\mathsf{z}.\mathsf{A}](\mathsf{w}.\mathsf{y}.\tau;\, \mathsf{p})$$

we will omit the most external parentheses in S-terms A, τ and p. However, we may write them for emphasis or in case of ambiguity.

▲

We now introduce the bind structures for Σ-types.

Definition 1.5.23 (Bind structures and local variables for Σ-types)**.** There are three kinds of *bind structures* for Σ-types.

- In a Σ-eliminator $\mathsf{ind}_{\Sigma}[\mathsf{z}.\mathsf{A}](\mathsf{w}.\mathsf{y}.\tau;\, \mathsf{p})$ there are two bind structures:
  - The expression $\mathsf{z}.\mathsf{A}$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{z}\}$.
  - The expression $\mathsf{w}.\mathsf{y}.\tau$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{w}, \mathsf{y}\}$.
- In a Σ-type expression $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{B}$, a *bind structure* is defined by the underlined parts, as indicated in the following expression:

  $$\sum\nolimits_{\underline{\mathsf{w}:\,\mathsf{A}}} \underline{\mathsf{B}}$$

  The set of *local variables* in the bind structure is defined to be $\{\mathsf{w}\}$.

▲

It is time to state the computation rule.

- Computation (S-context: Γ; S-term: g, a, b, A, B, C; Variable: w, y, z):

$$\frac{\Gamma, \mathsf{z}: \sum\limits_{\mathsf{w}:\mathsf{A}} \mathsf{B} \vdash \mathsf{C}: \mathcal{U}_i \qquad \Gamma, \mathsf{w}: \mathsf{A}, \mathsf{y}: \mathsf{B} \vdash \mathsf{g}: \mathsf{C}[\mathsf{z}/(\mathsf{w},\mathsf{y})] \qquad \Gamma \vdash \mathsf{a}: \mathsf{A} \qquad \Gamma \vdash \mathsf{b}: \mathsf{B}[\mathsf{w}/\mathsf{a}]}{\Gamma \vdash \mathsf{ind}_{\Sigma}[\mathsf{z}.\mathsf{C}](\mathsf{w}.\mathsf{y}.\mathsf{g};\, (\mathsf{a},\mathsf{b})) \equiv \mathsf{g}[\mathsf{w}/\mathsf{a}, \mathsf{y}/\mathsf{b}]: \mathsf{C}[\mathsf{z}/(\mathsf{a},\mathsf{b})]}\ \Sigma\text{-COMP}$$

  Informal reading. If the proof produced by the elimination rule is applied on the canonical pair (a, b), then this proof is just g instantiated with a and b.

  To understand the intuition behind the computation rule, it is useful to reinterpret the *elimination* rule in a procedural way. Given an arbitrary pair p, the elimination rule produces a proof for $\mathsf{C}[\mathsf{z}/\mathsf{p}]$ by "decomposing" p into a pair (r, t), for some r and t. Then, it passes r and t into g (since g can be seen as a function[14] that produces a proof for $\mathsf{C}[\mathsf{z}/(\mathsf{w},\mathsf{y})]$ given inputs w and y) to produce a proof for $\mathsf{C}[\mathsf{z}/(\mathsf{r},\mathsf{t})]$, or equivalently, $\mathsf{C}[\mathsf{z}/\mathsf{p}]$.

  Hence, if pair p is *already decomposed* as (a, b), then the proof produced by the elimination rule is actually g applied on a and b, as the computation rule suggests.

  In the beginning of Section 1.5 it was stated that a computation rule describes what happens when a constructor is given to an eliminator. In this case, a canonical pair (i.e. the constructor, because canonical pairs are produced by the introduction rule) is placed inside a Σ-eliminator.

HoTT does not state a uniqueness rule for Σ-types. However, there is a propositional uniqueness principle, see Lemma 1.6.1.

The elimination and computation rules can be encapsulated into an induction principle, as the following lemma shows.

Lemma 1.5.24 (Induction principle for Σ-types)**.** *Let $\mathsf{A}: \mathcal{U}_i$ be a type and $\mathsf{P}: \mathsf{A} \to \mathcal{U}_j$ a type family.*
If $\mathsf{C}: (\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}) \to \mathcal{U}_k$ is a type family and we have a function:

$$\mathsf{g}: \prod_{\mathsf{w}:\mathsf{A}} \prod_{\mathsf{y}:\mathsf{P}\ \mathsf{w}} \mathsf{C}\ (\mathsf{w},\mathsf{y})$$

[14] Apply twice the Π-INTRO rule on condition $\Gamma, \mathsf{w}: \mathsf{A}, \mathsf{y}: \mathsf{B} \vdash \mathsf{g}: \mathsf{C}[\mathsf{z}/(\mathsf{w},\mathsf{y})]$ to get the desired function.

Then, we can define a function:

$$\mathsf{f}\colon \prod_{\mathsf{p}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{C}\ \mathsf{p}$$

as:

$$\mathsf{f}\ (\mathsf{a},\mathsf{b}) :\equiv \mathsf{g}\ \mathsf{a}\ \mathsf{b}$$

where $\mathsf{a}\colon \mathsf{A}$ *and* $\mathsf{b}\colon \mathsf{P}\ \mathsf{a}$.

Proof. If we define the context:

$$\Gamma :\equiv \mathsf{A}\colon \mathcal{U}_i,\ \mathsf{P}\colon \mathsf{A} \to \mathcal{U}_j,\ \mathsf{C}\colon \left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right) \to \mathcal{U}_k,\ \mathsf{g}\colon \prod_{\mathsf{w}:\mathsf{A}} \prod_{\mathsf{y}:\mathsf{P}\ \mathsf{w}} \mathsf{C}\ (\mathsf{w},\mathsf{y})$$

then the lemma is claiming that there is an S-term f such that the following judgments are derivable:

$$\Gamma \vdash \mathsf{f}\colon \prod_{\mathsf{p}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{C}\ \mathsf{p}$$

$$\Gamma,\ \mathsf{a}\colon \mathsf{A},\ \mathsf{b}\colon \mathsf{P}\ \mathsf{a} \vdash \mathsf{f}\ (\mathsf{a},\mathsf{b}) \equiv \mathsf{g}\ \mathsf{a}\ \mathsf{b}\colon \mathsf{C}\ (\mathsf{a},\mathsf{b})$$

For the first judgment, suppose context Γ, and then define:

$$\mathsf{f} :\equiv \lambda \mathsf{p}\colon \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}.\ \mathsf{ind}_{\Sigma}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{y}.\mathsf{g}\ \mathsf{w}\ \mathsf{y};\ \mathsf{p})$$

Now, we check this f has the required type. Suppose $\mathsf{p}\colon \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ (i.e. we are augmenting our context with term p). We need to check:

$$\mathsf{ind}_{\Sigma}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{y}.\mathsf{g}\ \mathsf{w}\ \mathsf{y};\ \mathsf{p})\colon \mathsf{C}\ \mathsf{p}$$

We will use the Σ-ELIM rule, so we need to prove:

- Given $\mathsf{z}\colon \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$, term $\mathsf{C}\ \mathsf{z}$ is a type. Proof: apply Π-ELIM on C to get $\mathsf{C}\ \mathsf{z}\colon \mathcal{U}_k$.
- Given $\mathsf{w}\colon \mathsf{A}$ and $\mathsf{y}\colon \mathsf{P}\ \mathsf{w}$, term $\mathsf{g}\ \mathsf{w}\ \mathsf{y}\colon (\mathsf{C}\ \mathsf{z})[\mathsf{z}/(\mathsf{w},\mathsf{y})]$ holds. Proof: By two applications of the type family version of Π-ELIM (Lemma 1.5.15), we have $\mathsf{g}\ \mathsf{w}\ \mathsf{y}\colon \mathsf{C}\ (\mathsf{w},\mathsf{y})$. But $(\mathsf{C}\ \mathsf{z})[\mathsf{z}/(\mathsf{w},\mathsf{y})]$ denotes $\mathsf{C}\ (\mathsf{w},\mathsf{y})$.
- $\mathsf{p}\colon \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ holds. Proof: by assumption.

Hence, by Σ-ELIM we have:

$$\mathsf{ind}_{\Sigma}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{y}.\mathsf{g}\ \mathsf{w}\ \mathsf{y};\ \mathsf{p})\colon (\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{p}]$$

But $(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{p}]$ denotes $\mathsf{C}\ \mathsf{p}$.
Now, for the second judgment[15], we have by the Π-COMP and Σ-COMP rules:

$$\begin{aligned}
\mathsf{f}\ (\mathsf{a},\mathsf{b}) &\equiv (\lambda \mathsf{p}.\ \mathsf{ind}_{\Sigma}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{y}.\mathsf{g}\ \mathsf{w}\ \mathsf{y};\ \mathsf{p}))\ (\mathsf{a},\mathsf{b}) \\
&\equiv (\mathsf{ind}_{\Sigma}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{y}.\mathsf{g}\ \mathsf{w}\ \mathsf{y};\ \mathsf{p}))[\mathsf{p}/(\mathsf{a},\mathsf{b})] \\
&\equiv \mathsf{ind}_{\Sigma}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{y}.\mathsf{g}\ \mathsf{w}\ \mathsf{y};\ (\mathsf{a},\mathsf{b})) \\
&\equiv (\mathsf{g}\ \mathsf{w}\ \mathsf{y})[\mathsf{w}/\mathsf{a},\mathsf{y}/\mathsf{b}] \\
&\equiv \mathsf{g}\ \mathsf{a}\ \mathsf{b}
\end{aligned}$$

□

[15]The context in the second judgment is an extension of the context in the first judgment. Hence, by the weakening rule on page 10, function f can still be derived from the assumptions in the extended context, justifying its use inside the definitional equality judgment.

The induction principle reads (under the Curry-Howard correspondence):[16] "Let C be a unary predicate on dependent pairs. If we have a proof g that all canonical pairs (w,y) in $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ satisfy C, then we will have a proof f that all pairs p in $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ satisfy C".

But, from the point of view of functions, the induction principle reads: "To define a function on dependent pairs, define it on canonical pairs (a,b), and freely use terms a and b inside the function body".

When family C in the induction principle is constant, we have a *recursion principle.*

Lemma 1.5.25 (Recursion principle for Σ-types)**.** *Let $\mathsf{A}:\mathcal{U}_i$ be a type and $\mathsf{P}:\mathsf{A}\to\mathcal{U}_j$ a type family.*

If $\mathsf{D}:\mathcal{U}_k$ is a type and we have a function:

$$\mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}((\mathsf{P}\ \mathsf{w})\to\mathsf{D})$$

Then, we can define a function:

$$\mathsf{f}:\left(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right)\to\mathsf{D}$$

as:

$$\mathsf{f}\ (\mathsf{a},\mathsf{b}):\equiv \mathsf{g}\ \mathsf{a}\ \mathsf{b}$$

where $\mathsf{a}:\mathsf{A}$ and $\mathsf{b}:\mathsf{P}\ \mathsf{a}$.

Proof. Suppose the context. Then, instantiate the induction principle with the constant family:

$$\mathsf{C}:\equiv(\lambda\mathsf{z}.\ \mathsf{D}):\left(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right)\to\mathcal{U}_k$$

And simplify applications accordingly. □

What do we mean by "instantiate" in the proof of the recursion principle? We know that the induction principle is really claiming that the following judgments are derivable:

$$\Gamma\vdash \mathsf{f}:\prod_{\mathsf{p}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}}\mathsf{C}\ \mathsf{p}$$

$$\Gamma,\ \mathsf{a}:\mathsf{A},\ \mathsf{b}:\mathsf{P}\ \mathsf{a}\vdash \mathsf{f}\ (\mathsf{a},\mathsf{b})\equiv \mathsf{g}\ \mathsf{a}\ \mathsf{b}:\mathsf{C}\ (\mathsf{a},\mathsf{b})$$

where f is some function and Γ is context:

$$\mathsf{A}:\mathcal{U}_i,\ \mathsf{P}:\mathsf{A}\to\mathcal{U}_j,\ \mathsf{C}:\left(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right)\to\mathcal{U}_k,\ \mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}\prod_{\mathsf{y}:\mathsf{P}\ \mathsf{w}}\mathsf{C}\ (\mathsf{w},\mathsf{y})$$

If we apply four times the Π-INTRO rule to the first judgment, we get:

$$\vdash(\lambda\mathsf{A}.\ \lambda\mathsf{P}.\ \lambda\mathsf{C}.\ \lambda\mathsf{g}.\ \mathsf{f}):\prod_{\mathsf{A}:\mathcal{U}_i}\prod_{\mathsf{P}:\mathsf{A}\to\mathcal{U}_j}\prod_{\mathsf{C}:\left(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right)\to\mathcal{U}_k}\left(\prod_{\mathsf{w}:\mathsf{A}}\prod_{\mathsf{y}:\mathsf{P}\ \mathsf{w}}\mathsf{C}\ (\mathsf{w},\mathsf{y})\right)\to\left(\prod_{\mathsf{p}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}}\mathsf{C}\ \mathsf{p}\right)\tag{1.6}$$

Let us define:

$$\mathsf{T}_{\mathsf{I}}:\equiv\prod_{\mathsf{A}:\mathcal{U}_i}\prod_{\mathsf{P}:\mathsf{A}\to\mathcal{U}_j}\prod_{\mathsf{C}:\left(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right)\to\mathcal{U}_k}\left(\prod_{\mathsf{w}:\mathsf{A}}\prod_{\mathsf{y}:\mathsf{P}\ \mathsf{w}}\mathsf{C}\ (\mathsf{w},\mathsf{y})\right)\to\left(\prod_{\mathsf{p}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}}\mathsf{C}\ \mathsf{p}\right)$$

[16] The dependent function type can be interpreted as a universally quantified statement, see Remark 1.5.17.

Hence, the induction principle is claiming that the dependent function type $\mathsf{T_I}$ is inhabited. Remember that under the Curry-Howard correspondence, a dependent function type can be interpreted as a universally quantified statement. Hence, type $\mathsf{T_I}$ reads as the statement of the induction principle (Lemma 1.5.24). The fact that $\mathsf{T_I}$ is inhabited means that the induction principle has a proof or it is a true theorem.

Therefore, *a theorem stated in natural language is really a dependent function type*:[17] the domain of the function corresponds to the condition of the theorem, and the codomain of the function corresponds to the conclusion of the theorem. The statement "Theorem A is true or has a proof" means that type A is inhabited.

But we have not explained what we mean by "instantiate". Let us continue the example. Now define the context:

$$\Delta :\equiv \mathsf{A}:\mathcal{U}_i,\ \mathsf{P}:\mathsf{A}\to\mathcal{U}_j,\ \mathsf{D}:\mathcal{U}_k,\ \mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}((\mathsf{P}\ \mathsf{w})\to\mathsf{D})$$

which corresponds to the assumptions on the recursion principle (Lemma 1.5.25).

Then, by the weakening rule applied on judgment (1.6), we have:

$$\Delta \vdash (\lambda\mathsf{A}.\ \lambda\mathsf{P}.\ \lambda\mathsf{C}.\ \lambda\mathsf{g}.\ \mathsf{f}):\mathsf{T_I}$$

Therefore, by applying the Π-ELIM rule four times with inputs A, P, $(\lambda\mathsf{z}.\ \mathsf{D})$, and g, we get:

$$\Delta \vdash (\lambda\mathsf{A}.\ \lambda\mathsf{P}.\ \lambda\mathsf{C}.\ \lambda\mathsf{g}.\ \mathsf{f})\ \mathsf{A}\ \mathsf{P}\ (\lambda\mathsf{z}.\ \mathsf{D})\ \mathsf{g}:\prod_{\mathsf{p}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}}(\lambda\mathsf{z}.\ \mathsf{D})\ \mathsf{p}$$

where $(\lambda\mathsf{z}.\ \mathsf{D})$ is the constant type family as used in the proof for the recursion principle. This judgment simplifies to (after performing applications and simplifying notation):

$$\Delta \vdash \mathsf{f}[\mathsf{A}/\mathsf{A}][\mathsf{P}/\mathsf{P}][\mathsf{C}/(\lambda\mathsf{z}.\ \mathsf{D})][\mathsf{g}/\mathsf{g}]:\left(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right)\to\mathsf{D}$$

which says: "Under the assumptions in Δ, type $(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w})\to\mathsf{D}$ is inhabited". This is precisely the statement of the recursion principle.

In other words, to prove the recursion principle, we applied inputs:

$$\mathsf{A},\ \mathsf{P},\ (\lambda\mathsf{z}.\ \mathsf{D}),\ \mathsf{g}$$

to the *proof* $(\lambda\mathsf{A}.\ \lambda\mathsf{P}.\ \lambda\mathsf{C}.\ \lambda\mathsf{g}.\ \mathsf{f})$ *for the induction principle* (type $\mathsf{T_I}$). Hence, the statement "instantiate the induction principle" means: apply the appropriate inputs to the proof of the dependent function type $\mathsf{T_I}$, as this will specialize type $\mathsf{T_I}$ into type $(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w})\to\mathsf{D}$, which is the conclusion of the recursion principle.

In informal proofs, it will be common to say "by Lemma X we have ..." or "apply Lemma X to get...". All these expressions will mean "instantiate Lemma X..." as explained in here.

As an example on how the induction and recursion principles are used, we define the coordinate projection functions.

Lemma 1.5.26 (Projection functions for Σ-types)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *be a type and* $\mathsf{P}:\mathsf{A}\to\mathcal{U}_j$ *a type family.*

Then, the two coordinate projection *functions:*

$$\mathsf{pr}_1:\left(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right)\to\mathsf{A}$$
$$\mathsf{pr}_2:\prod_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}}\mathsf{P}\ (\mathsf{pr}_1\ \mathsf{q})$$

are defined as:

$$\mathsf{pr}_1\ (\mathsf{a},\mathsf{b}):\equiv \mathsf{a}$$
$$\mathsf{pr}_2\ (\mathsf{a},\mathsf{b}):\equiv \mathsf{b}$$

[17] This will be true most of the time, as there will be theorems without assumptions. On those cases, the theorem is simply a non-function type.

The special case in which P *is a constant family* $\mathsf{P} :\equiv (\lambda \mathsf{z}.\ \mathsf{D}) : \mathsf{A} \to \mathcal{U}_j$ *for some* $\mathsf{D} : \mathcal{U}_j$ *such that* z *does not occur free in* D*, the types of the projection functions simplify to:*

$$\mathsf{pr}_1 :\ (\mathsf{A} \times \mathsf{D}) \to \mathsf{A}$$
$$\mathsf{pr}_2 :\ (\mathsf{A} \times \mathsf{D}) \to \mathsf{D}$$

Proof. The first projection function pr_1 follows by the recursion principle when instantiated with the function:

$$\mathsf{g} :\equiv (\lambda(\mathsf{w}:\mathsf{A})(\mathsf{y}:\mathsf{P}\ \mathsf{w}).\ \mathsf{w}) : \prod_{\mathsf{w}:\mathsf{A}} ((\mathsf{P}\ \mathsf{w}) \to \mathsf{A})$$

Hence, by the recursion principle, we have:

$$\mathsf{pr}_1\ (\mathsf{a}, \mathsf{b}) \equiv (\lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \mathsf{w})\ \mathsf{a}\ \mathsf{b} \equiv (\lambda \mathsf{y}.\ \mathsf{a})\ \mathsf{b} \equiv \mathsf{a}$$

The second projection function pr_2 follows by the induction principle when instantiated with the family:

$$\mathsf{C} :\equiv (\lambda \mathsf{z}.\ \mathsf{P}\ (\mathsf{pr}_1\ \mathsf{z})) : \left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right) \to \mathcal{U}_j$$

and function:

$$\mathsf{g} :\equiv (\lambda(\mathsf{w}:\mathsf{A})(\mathsf{y}:\mathsf{P}\ \mathsf{w}).\ \mathsf{y}) : \prod_{\mathsf{w}:\mathsf{A}} \prod_{\mathsf{y}:\mathsf{P}\ \mathsf{w}} \mathsf{C}\ (\mathsf{w}, \mathsf{y})$$

Notice that g is well-typed, since:

$$\mathsf{C}\ (\mathsf{w}, \mathsf{y}) \equiv \mathsf{P}\ (\mathsf{pr}_1\ (\mathsf{w}, \mathsf{y})) \equiv \mathsf{P}\ \mathsf{w}$$

Hence, by the induction principle, we have:

$$\mathsf{pr}_2\ (\mathsf{a}, \mathsf{b}) \equiv (\lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \mathsf{y})\ \mathsf{a}\ \mathsf{b} \equiv (\lambda \mathsf{y}.\ \mathsf{y})\ \mathsf{b} \equiv \mathsf{b}$$

□

We introduce the following convention.

Convention 1.5.27. An S-term of the form:

$$\prod_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{C}\ \mathsf{q}$$

can be written as:

$$\prod_{(\mathsf{r},\mathsf{t}):\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{C}\ (\mathsf{r}, \mathsf{t})$$

Observe that this convention is justified by the induction principle for Σ-types, because the induction principle can prove statements of the form:

$$\prod_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{C}\ \mathsf{q}$$

by assuming that q has a canonical form, like (r, t), for some r and t.

Similarly, an S-term of the form:

$$\sum_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{C}\ \mathsf{q}$$

can be written as:

$$\sum_{(\mathsf{r},\mathsf{t}):\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{C}\ (\mathsf{r}, \mathsf{t})$$

▲

For example, following this convention, the second projection function can be written as:

$$\mathsf{pr}_2 : \prod_{(\mathsf{r},\mathsf{t}):\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{P}\ \mathsf{r}$$

since $\mathsf{P}\ (\mathsf{pr}_1\ (\mathsf{r},\mathsf{t})) \equiv \mathsf{P}\ \mathsf{r}$ by definition of the first projection function.

Another useful function will be the Σ_{map} function. This function represents a "functorial mapping" on Σ-types, i.e. if we have maps between coordinates, then we have a map between pairs.

Lemma 1.5.28 (Σ_{map} function)**.** *Let* $\mathsf{A} : \mathcal{U}_i$, $\mathsf{B} : \mathcal{U}_j$ *be types,* $\mathsf{P} : \mathsf{A} \to \mathcal{U}_k$, $\mathsf{Q} : \mathsf{B} \to \mathcal{U}_l$ *type families, and* $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ *a function.*

If we are given a function:

$$\mathsf{h} : \prod_{\mathsf{w}:\mathsf{A}} (\mathsf{P}\ \mathsf{w}) \to (\mathsf{Q}\ (\mathsf{f}\ \mathsf{w}))$$

then, we define the Σ_{map} *function:*

$$\Sigma_{\mathsf{map}}\ \mathsf{f}\ \mathsf{h} : \left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right) \to \left(\sum_{\mathsf{w}:\mathsf{B}} \mathsf{Q}\ \mathsf{w}\right)$$

as:

$$\Sigma_{\mathsf{map}}\ \mathsf{f}\ \mathsf{h}\ (\mathsf{a},\mathsf{b}) :\equiv (\mathsf{f}\ \mathsf{a},\ \mathsf{h}\ \mathsf{a}\ \mathsf{b})$$

where $\mathsf{a} : \mathsf{A}$ *and* $\mathsf{b} : \mathsf{P}\ \mathsf{a}$.

Proof. By the recursion principle for Σ-types, it is enough to define a function:

$$\mathsf{g} : \prod_{\mathsf{w}:\mathsf{A}} \left((\mathsf{P}\ \mathsf{w}) \to \left(\sum_{\mathsf{y}:\mathsf{B}} \mathsf{Q}\ \mathsf{y}\right)\right)$$

So, define:

$$\mathsf{g} :\equiv \lambda(\mathsf{w}:\mathsf{A})(\mathsf{q}:\mathsf{P}\ \mathsf{w}).\ (\mathsf{f}\ \mathsf{w},\ \mathsf{h}\ \mathsf{w}\ \mathsf{q})$$

Notice g is well typed, since $\mathsf{h}\ \mathsf{w}\ \mathsf{q} : \mathsf{Q}\ (\mathsf{f}\ \mathsf{w})$.

Hence, we have:

$$\Sigma_{\mathsf{map}}\ \mathsf{f}\ \mathsf{h}\ (\mathsf{a},\mathsf{b}) \equiv \mathsf{g}\ \mathsf{a}\ \mathsf{b} \equiv (\mathsf{f}\ \mathsf{a},\ \mathsf{h}\ \mathsf{a}\ \mathsf{b})$$

□

Before ending this section, we have a couple of remarks.

Remark 1.5.29 (Universe level for Σ-types)**.** To compute the universe level of $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{B}$, just take the maximum between the levels of A and B (for every $\mathsf{w} : \mathsf{A}$). The reasoning is similar to the one in Remark 1.5.16.

▲

Remark 1.5.30 (Logical interpretation of Σ-types)**.** Under the Curry-Howard correspondence, a Σ-type corresponds to an *existentially quantified statement* [18]. In other words, type $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{B}$ can be interpreted as: "There is a proof w for A such that B holds".

However, there is a fine point here. A Σ-type is a *constructive existential*. A constructive existential not only claims that certain object exists, but also provides the actual object whose existence is claimed. In more detail, given a pair $(\mathsf{a},\mathsf{b}) : \sum_{\mathsf{w}:\mathsf{A}} \mathsf{B}$, term a is the object claimed to exist, and b is a proof that object a satisfies proposition B. This means that pairs can be interpreted as proofs for the existential statement, as they *are* the required evidence.

To the contrary, a *classical existential* (as understood in first-order classical logic) claims that an object exists without necessarily constructing the object whose existence is claimed. This is why indirect proofs for existential claims are acceptable in classical logic.

In classical logic, an indirect proof for an existential claim has the following form:

- Assume that an object satisfying the claim *does not* exist.

- Prove intermediate steps.
- A contradiction is reached. Conclude that an object satisfying the claim *must* exist.

However, an indirect proof as above only shows that *some* object exists, but we do not know the specific object that does (i.e. we cannot point our finger to the particular object satisfying the claim). In classical logic, indirect proofs are sound because classical existentials are not required to "show" the object they claim.

We will see that HoTT is able to represent classical existentials as well, once we introduce the concept of (-1)-truncations in Section 1.11.1.

Finally, the product type $\mathsf{A} \times \mathsf{B}$ corresponds to *logical conjunction*. In other words, type $\mathsf{A} \times \mathsf{B}$ can be interpreted as: "Both A and B hold". Given a pair $(\mathsf{a}, \mathsf{b}) : \mathsf{A} \times \mathsf{B}$, term a will be a proof for A, and b will be a proof for B.

▲

1.5.3 Sum type

If A and B are types, then $\mathsf{A} + \mathsf{B}$ will denote the *sum type* (also called *coproduct type*) of A and B. Intuitively, whenever A and B are interpreted as collections, $\mathsf{A} + \mathsf{B}$ will be their disjoint union.

We state the formation rule for sum types.

- Formation (S-context: Γ; S-term: A, B):

$$\frac{\Gamma \vdash \mathsf{A} : \mathcal{U}_i \qquad \Gamma \vdash \mathsf{B} : \mathcal{U}_i}{\Gamma \vdash \mathsf{A} + \mathsf{B} : \mathcal{U}_i} \; +\text{-FORM}$$

  Informal reading. If A and B are types at level i, then $\mathsf{A} + \mathsf{B}$ is a type at level i.

The formation rule requires both types to be at the same universe level, but this is not necessary, as the following lemma shows.

Lemma 1.5.31. *Let* $\mathsf{A} : \mathcal{U}_i$ *and* $\mathsf{B} : \mathcal{U}_j$ *be types. Then,* $\mathsf{A} + \mathsf{B}$ *is a type at level* $\max\{i, j\}$.

Proof. The statement is claiming that the following judgment is derivable:

$$\mathsf{A} : \mathcal{U}_i, \;\; \mathsf{B} : \mathcal{U}_j \vdash \mathsf{A} + \mathsf{B} : \mathcal{U}_{\max\{i,j\}}$$

So, we suppose the context. To be able to use the +-FORM rule we have to prove:

- Term A is a type at level $\max\{i, j\}$. Proof: by assumption we have $\mathsf{A} : \mathcal{U}_i$. Hence, by the cumulative universe rule we have $\mathsf{A} : \mathcal{U}_{\max\{i,j\}}$.
- Term B is a type at level $\max\{i, j\}$. Proof: by assumption we have $\mathsf{B} : \mathcal{U}_j$. Hence, by the cumulative universe rule we have $\mathsf{B} : \mathcal{U}_{\max\{i,j\}}$.

□

We follow the next convention regarding parentheses.

Convention 1.5.32 (Parentheses)**.** In an S-term of the form $\mathsf{A} + \mathsf{B}$ we will *never* remove the most external parentheses in A and B. But, if A or B are sum types themselves, then their most external parentheses can be removed on the corresponding case.

For example, the S-term $\mathsf{A} + (\mathsf{C} + \mathsf{D})$ will be written as $\mathsf{A} + \mathsf{C} + \mathsf{D}$, but parentheses will be kept in $(\mathsf{C} \times \mathsf{D}) + \mathsf{B}$.

Notice that there is ambiguity in the expression $\mathsf{A} + \mathsf{C} + \mathsf{D}$, because it could be interpreted as $(\mathsf{A} + \mathsf{C}) + \mathsf{D}$ or $\mathsf{A} + (\mathsf{C} + \mathsf{D})$. However, it can be proved that the sum type is associative.[18] So, it does not matter how S-term $\mathsf{A} + \mathsf{C} + \mathsf{D}$ is interpreted. But, for the sake of resolving ambiguity, $(\mathsf{A} + \mathsf{C}) + \mathsf{D}$ will be the preferred interpretation.

Of course, we may add parentheses for emphasis or when there is ambiguity.

▲

We state the introduction and elimination rules. This is the first case where we have two introduction rules.

[18] We omit the proof, as this situation will never occur on this report.

- Introduction 1 (S-context: Γ; S-term: α, A, B):

$$\frac{\Gamma \vdash \mathsf{A} : \mathcal{U}_i \qquad \Gamma \vdash \mathsf{B} : \mathcal{U}_i \qquad \Gamma \vdash \alpha : \mathsf{A}}{\Gamma \vdash \mathsf{inl}\ \alpha : \mathsf{A} + \mathsf{B}}\ +\text{-INTRO-1}$$

  Informal reading. Let α be a term belonging to A. Then, we can construct a copy of term α, denoted as $(\mathsf{inl}\ \alpha)$, in type $\mathsf{A} + \mathsf{B}$.

  Here, inl is called *left injection*, because the copy it produces will live at the type on the *left side* of $\mathsf{A} + \mathsf{B}$.

- Introduction 2 (S-context: Γ; S-term: β, A, B):

$$\frac{\Gamma \vdash \mathsf{A} : \mathcal{U}_i \qquad \Gamma \vdash \mathsf{B} : \mathcal{U}_i \qquad \Gamma \vdash \beta : \mathsf{B}}{\Gamma \vdash \mathsf{inr}\ \beta : \mathsf{A} + \mathsf{B}}\ +\text{-INTRO-2}$$

  Informal reading. Let β be a term belonging to B. Then, we can construct a copy of term β, denoted as $(\mathsf{inr}\ \beta)$, in type $\mathsf{A} + \mathsf{B}$.

  Here, inr is called *right injection*, because the copy it produces will live at the type on the *right side* of $\mathsf{A} + \mathsf{B}$.

- Elimination (S-context: Γ; S-term: g, h, q, A, B, C; Variable: w, y, z):

$$\frac{\Gamma, \mathsf{z} : \mathsf{A} + \mathsf{B} \vdash \mathsf{C} : \mathcal{U}_i \qquad \Gamma, \mathsf{w} : \mathsf{A} \vdash \mathsf{g} : \mathsf{C}[\mathsf{z}/\mathsf{inl}\ \mathsf{w}] \qquad \Gamma, \mathsf{y} : \mathsf{B} \vdash \mathsf{h} : \mathsf{C}[\mathsf{z}/\mathsf{inr}\ \mathsf{y}] \qquad \Gamma \vdash \mathsf{q} : \mathsf{A} + \mathsf{B}}{\Gamma \vdash \mathsf{ind}_+[\mathsf{z}.\mathsf{C}](\mathsf{w}.\mathsf{g}; \mathsf{y}.\mathsf{h}; \mathsf{q}) : \mathsf{C}[\mathsf{z}/\mathsf{q}]}\ +\text{-ELIM}$$

  Informal reading. Condition $\Gamma, \mathsf{z} : \mathsf{A} + \mathsf{B} \vdash \mathsf{C} : \mathcal{U}_i$ reads: "Let C be a unary predicate on the disjoint union of A and B".

  Condition $\Gamma \vdash \mathsf{q} : \mathsf{A} + \mathsf{B}$ reads: "Let q be an arbitrary element in the disjoint union".

  Then, the rest reads: "To prove that predicate C holds for q (i.e. type $\mathsf{C}[\mathsf{z}/\mathsf{q}]$ in the conclusion), prove that predicate C holds for all copies on the left side (i.e. condition $\Gamma, \mathsf{w} : \mathsf{A} \vdash \mathsf{g} : \mathsf{C}[\mathsf{z}/\mathsf{inl}\ \mathsf{w}]$) *and* all copies on the right side (i.e. condition $\Gamma, \mathsf{y} : \mathsf{B} \vdash \mathsf{h} : \mathsf{C}[\mathsf{z}/\mathsf{inr}\ \mathsf{y}]$)".

  The intuition is that if C holds for all *canonical* elements of type $\mathsf{A} + \mathsf{B}$ (a canonical element is a term of the form $(\mathsf{inl}\ \mathsf{w})$ or $(\mathsf{inr}\ \mathsf{y})$), then it will hold for an arbitrary element q in type $\mathsf{A} + \mathsf{B}$, because q can be "decomposed" into one of the forms $(\mathsf{inl}\ \mathsf{w})$ for some w or $(\mathsf{inr}\ \mathsf{y})$ for some y, as this is the only way to build elements of $\mathsf{A} + \mathsf{B}$ in the first place, i.e. the only way to build q is by using one of the introduction rules.

  The $+$-eliminator term $\mathsf{ind}_+[\mathsf{z}.\mathsf{C}](\mathsf{w}.\mathsf{g}; \mathsf{y}.\mathsf{h}; \mathsf{q})$ is a *witness* for this proof by elimination. It remembers all the pieces: the predicate C, the proof g that C holds for all copies on the left side of $\mathsf{A} + \mathsf{B}$, the proof h that C holds for all copies on the right side of $\mathsf{A} + \mathsf{B}$, and the arbitrary element q.

  The dot notation $\mathsf{z}.\mathsf{C}$, $\mathsf{w}.\mathsf{g}$, and $\mathsf{y}.\mathsf{h}$ in term $\mathsf{ind}_+[\mathsf{z}.\mathsf{C}](\mathsf{w}.\mathsf{g}; \mathsf{y}.\mathsf{h}; \mathsf{q})$ denotes bind structures (see Definition 1.5.34 below).

Convention 1.5.33 (Parentheses)**.** Given an S-term of the form:

$$\mathsf{ind}_+[\mathsf{z}.\mathsf{A}](\mathsf{w}.\tau; \mathsf{y}.\nu; \mathsf{p})$$

we will omit the most external parentheses in A, τ, ν, and p. However, we may write them for emphasis or in case of ambiguity.

▲

We now introduce the bind structures for sum types.

Definition 1.5.34 (Bind structures and local variables for sum types)**.** In a $+$-eliminator:

$$\mathsf{ind}_+[\mathsf{z}.\mathsf{A}](\mathsf{w}.\tau; \mathsf{y}.\nu; \mathsf{p})$$

there are three *bind structures*:

- The expression $\mathsf{z}.\mathsf{A}$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{z}\}$.
- The expression $\mathsf{w}.\tau$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{w}\}$.
- The expression $\mathsf{y}.\nu$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{y}\}$.

▲

It is time to state the computation rules. Since we have two constructors (i.e. two introduction rules), there are two ways to give a constructor into a $+$-eliminator. Hence, we will have two computation rules.

- Computation 1 (S-context: Γ; S-term: α, g, h, A, B, C; Variable: w, y, z):

$$\frac{\Gamma, \mathsf{z}\colon \mathsf{A}+\mathsf{B} \vdash \mathsf{C}\colon \mathcal{U}_i \qquad \Gamma, \mathsf{w}\colon \mathsf{A} \vdash \mathsf{g}\colon \mathsf{C}[\mathsf{z}/\mathsf{inl}\ \mathsf{w}] \qquad \Gamma, \mathsf{y}\colon \mathsf{B} \vdash \mathsf{h}\colon \mathsf{C}[\mathsf{z}/\mathsf{inr}\ \mathsf{y}] \qquad \Gamma \vdash \alpha\colon \mathsf{A}}{\Gamma \vdash \mathsf{ind}_+[\mathsf{z}.\mathsf{C}](\mathsf{w}.\mathsf{g};\ \mathsf{y}.\mathsf{h};\ \mathsf{inl}\ \alpha) \equiv \mathsf{g}[\mathsf{w}/\alpha]\colon \mathsf{C}[\mathsf{z}/\mathsf{inl}\ \alpha]}\ +\text{-COMP-1}$$

  Informal reading. If the the proof produced by the elimination rule is applied on the left injection $(\mathsf{inl}\ \alpha)$, then this proof is just g instantiated with α.

  To understand the intuition behind the computation rule, it is useful to reinterpret the *elimination* rule in a procedural way. Given arbitrary $\mathsf{q}\colon \mathsf{A}+\mathsf{B}$, the elimination rule produces a proof for $\mathsf{C}[\mathsf{z}/\mathsf{q}]$ by "decomposing" q into one of the forms $(\mathsf{inl}\ \mathsf{r})$ or $(\mathsf{inr}\ \mathsf{t})$, for some r and t. Then, it passes r or t into g or h, respectively, to produce a proof for one of the statements: $\mathsf{C}[\mathsf{z}/\mathsf{inl}\ \mathsf{r}]$ or $\mathsf{C}[\mathsf{z}/\mathsf{inr}\ \mathsf{t}]$ (both of them denoting statement $\mathsf{C}[\mathsf{z}/\mathsf{q}]$).

  Hence, if $\mathsf{q}\colon \mathsf{A}+\mathsf{B}$ is *already decomposed* as $(\mathsf{inl}\ \alpha)$, the proof produced by the elimination rule is actually g applied on α, as the computation rule suggests.

- Computation 2 (S-context: Γ; S-term: β, g, h, A, B, C; Variable: w, y, z):

$$\frac{\Gamma, \mathsf{z}\colon \mathsf{A}+\mathsf{B} \vdash \mathsf{C}\colon \mathcal{U}_i \qquad \Gamma, \mathsf{w}\colon \mathsf{A} \vdash \mathsf{g}\colon \mathsf{C}[\mathsf{z}/\mathsf{inl}\ \mathsf{w}] \qquad \Gamma, \mathsf{y}\colon \mathsf{B} \vdash \mathsf{h}\colon \mathsf{C}[\mathsf{z}/\mathsf{inr}\ \mathsf{y}] \qquad \Gamma \vdash \beta\colon \mathsf{B}}{\Gamma \vdash \mathsf{ind}_+[\mathsf{z}.\mathsf{C}](\mathsf{w}.\mathsf{g};\ \mathsf{y}.\mathsf{h};\ \mathsf{inr}\ \beta) \equiv \mathsf{h}[\mathsf{y}/\beta]\colon \mathsf{C}[\mathsf{z}/\mathsf{inr}\ \beta]}\ +\text{-COMP-2}$$

  Informal reading. The idea is similar to the computation 1 rule, but this time we work on right injections, and we use proof h instead of g.

HoTT does not state a uniqueness rule for sum types. Also, the elimination and computation rules can be encapsulated into an induction principle.

Lemma 1.5.35 (Induction principle for sum types)**.** *Let* $\mathsf{A}\colon \mathcal{U}_i$ *and* $\mathsf{B}\colon \mathcal{U}_j$ *be types.*
If $\mathsf{C}\colon (\mathsf{A}+\mathsf{B}) \to \mathcal{U}_k$ *is a type family, and we have two functions:*

$$\mathsf{g}\colon \prod_{\mathsf{w}:\mathsf{A}} \mathsf{C}\ (\mathsf{inl}\ \mathsf{w})$$
$$\mathsf{h}\colon \prod_{\mathsf{y}:\mathsf{B}} \mathsf{C}\ (\mathsf{inr}\ \mathsf{y})$$

Then, we can define a function:

$$\mathsf{f}\colon \prod_{\mathsf{q}:\mathsf{A}+\mathsf{B}} \mathsf{C}\ \mathsf{q}$$

as:

$$\mathsf{f}\ (\mathsf{inl}\ \mathsf{l}) :\equiv \mathsf{g}\ \mathsf{l}$$
$$\mathsf{f}\ (\mathsf{inr}\ \mathsf{r}) :\equiv \mathsf{h}\ \mathsf{r}$$

where $\mathsf{l}\colon \mathsf{A}$ *and* $\mathsf{r}\colon \mathsf{B}$.

Proof. If we define the context:

$$\Gamma :\equiv \mathsf{A}:\mathcal{U}_i,\ \mathsf{B}:\mathcal{U}_j,\ \mathsf{C}:\ (\mathsf{A}+\mathsf{B})\to\mathcal{U}_k,\ \mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}\mathsf{C}\ (\mathsf{inl}\ \mathsf{w}),\ \mathsf{h}:\prod_{\mathsf{y}:\mathsf{B}}\mathsf{C}\ (\mathsf{inr}\ \mathsf{y})$$

then, the lemma is claiming that there is an S-term f such that the following judgments are derivable:

$$\Gamma\vdash \mathsf{f}:\prod_{\mathsf{q}:\mathsf{A}+\mathsf{B}}\mathsf{C}\ \mathsf{q}$$
$$\Gamma,\mathsf{l}:\mathsf{A}\vdash \mathsf{f}\ (\mathsf{inl}\ \mathsf{l})\equiv \mathsf{g}\ \mathsf{l}:\mathsf{C}\ (\mathsf{inl}\ \mathsf{l})$$
$$\Gamma,\mathsf{r}:\mathsf{B}\vdash \mathsf{f}\ (\mathsf{inr}\ \mathsf{r})\equiv \mathsf{h}\ \mathsf{r}:\mathsf{C}\ (\mathsf{inr}\ \mathsf{r})$$

For the first judgment, suppose the context and then define:

$$\mathsf{f}:\equiv \lambda\mathsf{q}:\mathsf{A}+\mathsf{B}.\ \mathsf{ind}_+[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{g}\ \mathsf{w};\ \mathsf{y}.\mathsf{h}\ \mathsf{y};\ \mathsf{q})$$

Now, we check this f has the required type. Suppose $\mathsf{q}:\mathsf{A}+\mathsf{B}$. We need to check:

$$\mathsf{ind}_+[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{g}\ \mathsf{w};\ \mathsf{y}.\mathsf{h}\ \mathsf{y};\ \mathsf{q}):\mathsf{C}\ \mathsf{q}$$

We will use the +-ELIM rule, so we need to prove:

- Given $\mathsf{z}:\mathsf{A}+\mathsf{B}$, term $\mathsf{C}\ \mathsf{z}$ is a type. Proof: apply Π-ELIM to get $\mathsf{C}\ \mathsf{z}:\mathcal{U}_k$.
- Given $\mathsf{w}:\mathsf{A}$, check $\mathsf{g}\ \mathsf{w}:(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{inl}\ \mathsf{w}]$. Proof: apply the type family version of Π-ELIM to get $\mathsf{g}\ \mathsf{w}:\mathsf{C}\ (\mathsf{inl}\ \mathsf{w})$. But $(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{inl}\ \mathsf{w}]$ denotes $\mathsf{C}\ (\mathsf{inl}\ \mathsf{w})$.
- Given $\mathsf{y}:\mathsf{B}$, check $\mathsf{h}\ \mathsf{y}:(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{inr}\ \mathsf{y}]$. Proof: apply the type family version of Π-ELIM to get $\mathsf{h}\ \mathsf{y}:\mathsf{C}\ (\mathsf{inr}\ \mathsf{y})$. But $(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{inr}\ \mathsf{y}]$ denotes $\mathsf{C}\ (\mathsf{inr}\ \mathsf{y})$.
- $\mathsf{q}:\mathsf{A}+\mathsf{B}$ holds. Proof: by assumption.

Hence, we have:

$$\mathsf{ind}_+[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{g}\ \mathsf{w};\ \mathsf{y}.\mathsf{h}\ \mathsf{y};\ \mathsf{q}):(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{q}]$$

But $(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{q}]$ denotes $\mathsf{C}\ \mathsf{q}$.
Now, for the second judgment, we have by computation rules:

$$\begin{aligned}\mathsf{f}\ (\mathsf{inl}\ \mathsf{l}) &\equiv (\lambda\mathsf{q}.\ \mathsf{ind}_+[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{g}\ \mathsf{w};\ \mathsf{y}.\mathsf{h}\ \mathsf{y};\ \mathsf{q}))\ (\mathsf{inl}\ \mathsf{l})\\ &\equiv (\mathsf{ind}_+[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{g}\ \mathsf{w};\ \mathsf{y}.\mathsf{h}\ \mathsf{y};\ \mathsf{q}))[\mathsf{q}/\mathsf{inl}\ \mathsf{l}]\\ &\equiv \mathsf{ind}_+[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{g}\ \mathsf{w};\ \mathsf{y}.\mathsf{h}\ \mathsf{y};\ \mathsf{inl}\ \mathsf{l})\\ &\equiv (\mathsf{g}\ \mathsf{w})[\mathsf{w}/\mathsf{l}]\\ &\equiv \mathsf{g}\ \mathsf{l}\end{aligned}$$

And for the third judgment:

$$\begin{aligned}\mathsf{f}\ (\mathsf{inr}\ \mathsf{r}) &\equiv (\lambda\mathsf{q}.\ \mathsf{ind}_+[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{g}\ \mathsf{w};\ \mathsf{y}.\mathsf{h}\ \mathsf{y};\ \mathsf{q}))\ (\mathsf{inr}\ \mathsf{r})\\ &\equiv (\mathsf{ind}_+[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{g}\ \mathsf{w};\ \mathsf{y}.\mathsf{h}\ \mathsf{y};\ \mathsf{q}))[\mathsf{q}/\mathsf{inr}\ \mathsf{r}]\\ &\equiv \mathsf{ind}_+[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{w}.\mathsf{g}\ \mathsf{w};\ \mathsf{y}.\mathsf{h}\ \mathsf{y};\ \mathsf{inr}\ \mathsf{r})\\ &\equiv (\mathsf{h}\ \mathsf{y})[\mathsf{y}/\mathsf{r}]\\ &\equiv \mathsf{h}\ \mathsf{r}\end{aligned}$$

□

The induction principle reads (under the Curry-Howard correspondence): "Let C be a unary predicate on a disjoint union. If we have proofs g and h expressing that all canonical elements $(\mathsf{inl}\ \mathsf{l})$ and $(\mathsf{inr}\ \mathsf{r})$ in $\mathsf{A}+\mathsf{B}$ satisfy C, then we will have a proof f that all elements q in $\mathsf{A}+\mathsf{B}$ satisfy C".

But from the point of view of functions, the induction principle reads: "To define a function on a sum type, define it by cases on canonical elements $(\mathsf{inl}\ \mathsf{l})$ and $(\mathsf{inr}\ \mathsf{r})$, and freely use terms l and r inside the function body of the respective equation".

When family C in the induction principle is constant, we have a *recursion principle*.

Lemma 1.5.36 (Recursion principle for sum types)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *and* $\mathsf{B}:\mathcal{U}_j$ *be types.*

If $\mathsf{D}:\mathcal{U}_k$ *is a type and we have two functions:*

$$\mathsf{g}:\mathsf{A}\to\mathsf{D}$$
$$\mathsf{h}:\mathsf{B}\to\mathsf{D}$$

Then, we can define a function:

$$\mathsf{f}:(\mathsf{A}+\mathsf{B})\to\mathsf{D}$$

as:

$$\mathsf{f}\ (\mathsf{inl}\ \mathsf{l}) :\equiv \mathsf{g}\ \mathsf{l}$$
$$\mathsf{f}\ (\mathsf{inr}\ \mathsf{r}) :\equiv \mathsf{h}\ \mathsf{r}$$

where $\mathsf{l}:\mathsf{A}$ *and* $\mathsf{r}:\mathsf{B}$.

Proof. Instantiate the induction principle with the constant family:

$$\mathsf{C} :\equiv (\lambda \mathsf{z}.\ \mathsf{D}):\ (\mathsf{A}+\mathsf{B})\to\mathcal{U}_k$$

And simplify applications accordingly.

□

Before ending this section, we have a couple of remarks.

Remark 1.5.37 (Universe level for sum types)**.** To compute the universe level of $\mathsf{A}+\mathsf{B}$, take the maximum of the levels of A and B. This is a direct consequence of Lemma 1.5.31.

▲

Remark 1.5.38 (Logical interpretation of sum types)**.** Under the Curry-Howard correspondence, a sum type corresponds to *logical disjunction* [18]. In other words, type $\mathsf{A}+\mathsf{B}$ can be interpreted as: "Either A or B holds".

However, a sum type is a *constructive disjunction*. Constructive disjunctions not only claim that one of two cases holds, but also indicate which case actually happens. For example, when we have a term $\mathsf{inl}\ \alpha:\mathsf{A}+\mathsf{B}$, this is a proof for either A or B, but we *also* know that A was the case that actually happened, because we are given a proof α for A injected on the *left* side of $\mathsf{A}+\mathsf{B}$. The reasoning is similar for terms $\mathsf{inr}\ \beta:\mathsf{A}+\mathsf{B}$.

Instead, a *classical disjunction* (as understood in first-order classical logic) claims that one of two cases holds without necessarily indicating the case that actually happens. This is why indirect proofs for classical disjunctions are acceptable in classical logic.

In classical logic, an indirect proof for a disjunction has the following form:

- Assume that *neither* of the cases is true.
- Prove intermediate steps.
- A contradiction is reached. Conclude that *one* of the cases must be true.

However, an indirect proof as above only shows that one of the cases holds, but we cannot point our finger to the actual case that holds.

Let us do an example to understand the difference between a classical disjunction and a constructive one. Suppose P is the statement of some mathematical conjecture which has not been proved or disproved. Then, in classical logic we have that:

$$\text{“}\mathsf{P} \text{ is true or false”} \tag{1.7}$$

is a true statement because we are allowed to use an indirect proof, i.e. assuming “P is neither true nor false” is trivially a contradiction. In other words, we are not required to indicate the case that holds when proving a disjunction.

Instead, in a constructive logic like HoTT, if we want to prove (1.7), we have to either construct a proof for P or disprove[19] P, since we are forced to indicate the case that actually holds (i.e. we cannot use an indirect proof because this will “hide” the case).

We will see that HoTT is able to represent classical disjunctions as well, once we introduce the concept of (-1)-truncations in Section 1.11.1.

▲

1.5.4 The empty type $\mathbb{0}$

The *empty type* $\mathbb{0}$ will represent a collection without elements. Hence, it will not have constructors.

We have the following inference rules.

- Formation (S-context: Γ):

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathbb{0} : \mathcal{U}_i}\ \mathbb{0}\text{-FORM}$$

  Informal reading. $\mathbb{0}$ is a type at an arbitrary level i.

  This rule means that type $\mathbb{0}$ is at every universe level.

- Elimination (S-context: Γ; S-term: α, C; Variable: z):

$$\frac{\Gamma, \mathsf{z} : \mathbb{0} \vdash \mathsf{C} : \mathcal{U}_i \qquad \Gamma \vdash \alpha : \mathbb{0}}{\Gamma \vdash \mathsf{ind}_{\mathbb{0}}[\mathsf{z}.\mathsf{C}](\alpha) : \mathsf{C}[\mathsf{z}/\alpha]}\ \mathbb{0}\text{-ELIM}$$

  Informal reading. Condition $\Gamma, \mathsf{z} : \mathbb{0} \vdash \mathsf{C} : \mathcal{U}_i$ reads: “Let C be a unary predicate on the empty type”.

  Then, the rest reads: “If we are able to construct an element α on the empty type, then we are in an inconsistent state (because $\mathbb{0}$ is *empty*!) and we can conclude anything. In particular, we can conclude that C holds for α”.

  The elimination rule captures the *ex falso quodlibet* rule in logic, which states that once a contradiction is reached, any proposition follows. Hence, we can imagine type $\mathbb{0}$ as representing a contradiction.

  The $\mathbb{0}$-eliminator term $\mathsf{ind}_{\mathbb{0}}[\mathsf{z}.\mathsf{C}](\alpha)$ is a *witness* for this proof by elimination. It remembers all the pieces: the predicate C, and the (impossible) element α.

  The dot notation $\mathsf{z}.\mathsf{C}$ in term $\mathsf{ind}_{\mathbb{0}}[\mathsf{z}.\mathsf{C}](\alpha)$ denotes a bind structure (see Definition 1.5.40 below).

Since there are no introduction rules, there are no terms to be placed inside a $\mathbb{0}$-eliminator. Hence, there are no computation rules. Similarly, there are no uniqueness rules.

Convention 1.5.39 (Parentheses)**.** Given an S-term of the form:

$$\mathsf{ind}_{\mathbb{0}}[\mathsf{z}.\mathsf{A}](\tau)$$

we will omit the most external parentheses in S-terms A and τ. However, we may write them for emphasis or in case of ambiguity.

▲

We now introduce the bind structures for $\mathbb{0}$.

[19] Section 1.5.4 will explain how falsity is modeled in HoTT.

Definition 1.5.40 (Bind structures and local variables for the empty type)**.** In a $\mathbb{0}$-eliminator:

$$\mathsf{ind}_{\mathbb{0}}[\mathsf{z}.\mathsf{A}](\tau)$$

The expression $\mathsf{z}.\mathsf{A}$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{z}\}$.

▲

We have the following induction principle for the empty type.

Lemma 1.5.41 (Induction principle for the empty type)**.** *If $\mathsf{C}\colon \mathbb{0} \to \mathcal{U}_i$ is a type family. Then, the following type is inhabited:*

$$\prod_{\mathsf{q}:\mathbb{0}} \mathsf{C}\ \mathsf{q}$$

Proof. The lemma is claiming that there is an S-term f such that the following judgment is derivable:

$$\mathsf{C}\colon \mathbb{0} \to \mathcal{U}_i \vdash \mathsf{f}\colon \prod_{\mathsf{q}:\mathbb{0}} \mathsf{C}\ \mathsf{q}$$

So, suppose the context and then define:

$$\mathsf{f} :\equiv \lambda \mathsf{q}\colon \mathbb{0}.\ \mathsf{ind}_{\mathbb{0}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{q})$$

We need to check this f has the required type. Suppose $\mathsf{q}\colon \mathbb{0}$. We need to check:

$$\mathsf{ind}_{\mathbb{0}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{q})\colon \mathsf{C}\ \mathsf{q}$$

We will use the $\mathbb{0}$-ELIM rule, so we need to prove:

- Given $\mathsf{z}\colon \mathbb{0}$, term $\mathsf{C}\ \mathsf{z}$ is a type. Proof: apply Π-ELIM to get $\mathsf{C}\ \mathsf{z}\colon \mathcal{U}_i$.
- Check $\mathsf{q}\colon \mathbb{0}$. Proof: by assumption.

Hence, we have:

$$\mathsf{ind}_{\mathbb{0}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{q})\colon (\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{q}]$$

But $(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{q}]$ denotes $\mathsf{C}\ \mathsf{q}$.

□

The induction principle reads (under the Curry-Howard correspondence): "From a contradiction, we can conclude any predicate C whatsoever" or "Any element of $\mathbb{0}$ satisfies any predicate C whatsoever", which is vacuously true since there are no elements in $\mathbb{0}$.

When family C in the induction principle is constant, we have a *recursion principle*.

Lemma 1.5.42 (Recursion principle for the empty type)**.** *If $\mathsf{D}\colon \mathcal{U}_i$ is a type. Then, the following type is inhabited:*

$$\mathbb{0} \to \mathsf{D}$$

Proof. Instantiate the induction principle with the constant family:

$$\mathsf{C} :\equiv (\lambda \mathsf{z}.\ \mathsf{D})\colon \mathbb{0} \to \mathcal{U}_i$$

And simplify applications accordingly.

□

Intuitively, the recursion principle states that there is always a function from the empty collection into an arbitrary collection. In fact, we should be able to prove that this function is unique (see Lemma 1.8.2).

To express falsity in HoTT, we introduce the *negation operator*, which can be defined with the aid of the empty type.

Definition 1.5.43 (Negation operator $\neg$)**.** Let $\mathsf{A} : \mathcal{U}_i$ be a type. Then, type:

$$\mathsf{A} \to \mathbb{0}$$

will be denoted as $\neg\mathsf{A}$. We read $\neg\mathsf{A}$ as "not A" or "A is not inhabited".

▲

Before ending this section, we have a remark.

Remark 1.5.44 (Logical interpretation of $\mathbb{0}$)**.** Under the Curry-Howard correspondence, the empty type corresponds to the "always false" proposition $\bot$ or *contradiction* [18]. Hence, proposition $\neg\mathsf{A}$ (denoting $\mathsf{A} \to \mathbb{0}$) means that "$\mathsf{A}$ is refutable", because a contradiction is reached if we assume A.

▲

1.5.5 The unit type $\mathbb{1}$

The *unit type* $\mathbb{1}$ will represent a collection with exactly one element.

We state the formation, introduction, and elimination rules.

- Formation (S-context: Γ):

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathbb{1} : \mathcal{U}_i}\ \mathbb{1}\text{-FORM}$$

  Informal reading. $\mathbb{1}$ is a type at an arbitrary level i.

  This rule means that type $\mathbb{1}$ is at every universe level.

- Introduction (S-context: Γ):

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \star : \mathbb{1}}\ \mathbb{1}\text{-INTRO}$$

  Informal reading. $\star$ is an element of $\mathbb{1}$.

- Elimination (S-context: Γ; S-term: g, q, C; Variable: z):

$$\frac{\Gamma, \mathsf{z} : \mathbb{1} \vdash \mathsf{C} : \mathcal{U}_i \qquad \Gamma \vdash \mathsf{g} : \mathsf{C}[\mathsf{z}/\star] \qquad \Gamma \vdash \mathsf{q} : \mathbb{1}}{\Gamma \vdash \mathsf{ind}_{\mathbb{1}}[\mathsf{z}.\mathsf{C}](\mathsf{g}; \mathsf{q}) : \mathsf{C}[\mathsf{z}/\mathsf{q}]}\ \mathbb{1}\text{-ELIM}$$

  Informal reading. Condition $\Gamma, \mathsf{z} : \mathbb{1} \vdash \mathsf{C} : \mathcal{U}_i$ reads: "Let C be a unary predicate on the unit type".

  Condition $\Gamma \vdash \mathsf{q} : \mathbb{1}$ reads: "Let q be an arbitrary element in the unit type".

  Then, the rest reads: "To prove that predicate C holds for q (i.e. type $\mathsf{C}[\mathsf{z}/\mathsf{q}]$ in the conclusion), prove that predicate C holds for $\star$ (i.e. condition $\Gamma \vdash \mathsf{g} : \mathsf{C}[\mathsf{z}/\star]$)".

  The intuition is that if C holds for $\star$, then it will hold for some arbitrary element q in $\mathbb{1}$, because q *must* be $\star$, as the only way to build elements in $\mathbb{1}$ is through the introduction rule.

  The $\mathbb{1}$-eliminator term $\mathsf{ind}_{\mathbb{1}}[\mathsf{z}.\mathsf{C}](\mathsf{g}; \mathsf{q})$ is a *witness* for this proof by elimination. It remembers all the pieces: the predicate C, the proof g that C holds for $\star$, and the arbitrary element q.

  The dot notation $\mathsf{z}.\mathsf{C}$ in term $\mathsf{ind}_{\mathbb{1}}[\mathsf{z}.\mathsf{C}](\mathsf{g}; \mathsf{p})$ denotes a bind structure (see Definition 1.5.46 below).

Convention 1.5.45 (Parentheses)**.** Given an S-term of the form:

$$\mathsf{ind}_{\mathbb{1}}[\mathsf{z}.\mathsf{A}](\tau; \nu)$$

we will omit the most external parentheses in S-terms A, τ, and ν. However, we may write them for emphasis or in case of ambiguity.

▲

We now introduce the bind structures for the unit type.

Definition 1.5.46 (Bind structures and local variables for the unit type)**.** In a $\mathbb{1}$-eliminator:

$$\mathsf{ind}_{\mathbb{1}}[\mathsf{z}.\mathsf{A}](\tau;\nu)$$

The expression $\mathsf{z}.\mathsf{A}$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{z}\}$.

▲

Now the computation rule.

- Computation (S-context: Γ; S-term: g, C; Variable: z):

$$\frac{\Gamma, \mathsf{z}:\mathbb{1} \vdash \mathsf{C}:\mathcal{U}_i \qquad \Gamma \vdash \mathsf{g}:\mathsf{C}[\mathsf{z}/\star]}{\Gamma \vdash \mathsf{ind}_{\mathbb{1}}[\mathsf{z}.\mathsf{C}](\mathsf{g};\star) \equiv \mathsf{g}:\mathsf{C}[\mathsf{z}/\star]}\ \mathbb{1}\text{-COMP}$$

Informal reading. If the proof produced by the elimination rule is applied on $\star$, then this proof must be g.

To understand the intuition behind the computation rule, it is useful to reinterpret the *elimination* rule in a procedural way. Given arbitrary $\mathsf{q}:\mathbb{1}$, the elimination rule produces a proof for $\mathsf{C}[\mathsf{z}/\mathsf{q}]$ by "decomposing" q into the form $\star$. Then, it returns the proof g, which is a proof for $\mathsf{C}[\mathsf{z}/\star]$, or equivalently, $\mathsf{C}[\mathsf{z}/\mathsf{q}]$.

Hence, if $\mathsf{q}:\mathbb{1}$ is *already decomposed* as $\star$, the proof produced by the elimination rule is actually g, as the computation rule suggests.

HoTT does not state a uniqueness rule for the unit type, but it has a propositional uniqueness principle (see Lemma 1.6.2).

Also, the elimination and computation rules can be encapsulated into an induction principle.

Lemma 1.5.47 (Induction principle for the unit type)**.** *If $\mathsf{C}:\mathbb{1}\to\mathcal{U}_i$ is a type family and we have a term $\mathsf{g}:\mathsf{C}\,\star$, then we can define a function:*

$$\mathsf{f}:\prod_{\mathsf{q}:\mathbb{1}}\mathsf{C}\,\mathsf{q}$$

as:

$$\mathsf{f}\,\star :\equiv \mathsf{g}$$

Proof. The pattern should be clear by now. Just define:

$$\mathsf{f} :\equiv \lambda\mathsf{q}:\mathbb{1}.\ \mathsf{ind}_{\mathbb{1}}[\mathsf{z}.\mathsf{C}\,\mathsf{z}](\mathsf{g};\mathsf{q})$$

and then check that f is well-typed by using the $\mathbb{1}$-ELIM rule.

The equation $\mathsf{f}\,\star \equiv \mathsf{g}$ follows by unfolding definitions and computation.

□

The induction principle reads (under the Curry-Howard correspondence): "Let C be a unary predicate on the unit type. To prove that C holds for all elements in $\mathbb{1}$, just prove that C holds for $\star$".

But from the point of view of functions, the induction principle reads: "To define a function on the unit type, it is enough to define it on $\star$".

When family C in the induction principle is constant, we have a *recursion principle*.

Lemma 1.5.48 (Recursion principle for the unit type)**.** *If $\mathsf{D}:\mathcal{U}_i$ is a type and we have a term $\mathsf{g}:\mathsf{D}$, then we can define a function:*

$$\mathsf{f}:\mathbb{1}\to\mathsf{D}$$

as:

$$\mathsf{f}\,\star :\equiv \mathsf{g}$$

Proof. Instantiate the induction principle with the constant family:

$$\mathsf{C} :\equiv (\lambda\mathsf{z}.\ \mathsf{D}):\mathbb{1}\to\mathcal{U}_i$$

And simplify applications accordingly.

□

Before ending this section, we have the following remark.

Remark 1.5.49 (Logical interpretation of the unit type)**.** Under the Curry-Howard correspondence, the unit type corresponds to the "always true" proposition $\top$ or *tautology* [18].

▲

1.5.6 The natural number type $\mathbb{N}$

Type $\mathbb{N}$ will correspond to the collection of natural numbers.

We state the formation and introduction rules.

- Formation (S-context: Γ):

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathbb{N} : \mathcal{U}_i}\ \mathbb{N}\text{-FORM}$$

  Informal reading. $\mathbb{N}$ is a type at an arbitrary universe level i.

  This rule means that type $\mathbb{N}$ is at every universe level.

- Introduction 1 (S-context: Γ):

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathsf{0} : \mathbb{N}}\ \mathbb{N}\text{-INTRO-1}$$

  Informal reading. The *zero* term $\mathsf{0}$ is a natural number.

- Introduction 2 (S-context: Γ; S-term: n):

$$\frac{\Gamma \vdash \mathsf{n} : \mathbb{N}}{\Gamma \vdash \mathsf{succ}\ \mathsf{n} : \mathbb{N}}\ \mathbb{N}\text{-INTRO-2}$$

  Informal reading. If n is a natural number, then its *successor* $(\mathsf{succ}\ \mathsf{n})$ is a natural number.

Notice we have two introduction rules (i.e. two constructors), one for zero and another one for the successor. The elimination rule will encode the idea that these two constructors are the only way to get natural numbers.

- Elimination (S-context: Γ; S-term: g, h, n, C; Variable: w, z):

$$\frac{\Gamma, \mathsf{z} : \mathbb{N} \vdash \mathsf{C} : \mathcal{U}_i \qquad \Gamma \vdash \mathsf{g} : \mathsf{C}[\mathsf{z}/\mathsf{0}] \qquad \Gamma, \mathsf{z} : \mathbb{N}, \mathsf{w} : \mathsf{C} \vdash \mathsf{h} : \mathsf{C}[\mathsf{z}/\mathsf{succ}\ \mathsf{z}] \qquad \Gamma \vdash \mathsf{n} : \mathbb{N}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}; \mathsf{n}) : \mathsf{C}[\mathsf{z}/\mathsf{n}]}\ \mathbb{N}\text{-ELIM}$$

  Informal reading. Condition $\Gamma, \mathsf{z} : \mathbb{N} \vdash \mathsf{C} : \mathcal{U}_i$ reads: "Let C be a unary predicate on natural numbers".

  Condition $\Gamma \vdash \mathsf{n} : \mathbb{N}$ reads: "Let n be an arbitrary natural number".

  Then, the rest reads: "To prove that predicate C holds for n (i.e. type $\mathsf{C}[\mathsf{z}/\mathsf{n}]$ in the conclusion), prove:

  – Predicate C holds for $\mathsf{0}$, i.e. condition $\Gamma \vdash \mathsf{g} : \mathsf{C}[\mathsf{z}/\mathsf{0}]$.
  – If $\mathsf{z} : \mathbb{N}$ satisfies C, then $(\mathsf{succ}\ \mathsf{z})$ also satisfies C, i.e. condition $\Gamma, \mathsf{z} : \mathbb{N}, \mathsf{w} : \mathsf{C} \vdash \mathsf{h} : \mathsf{C}[\mathsf{z}/\mathsf{succ}\ \mathsf{z}]$.
    Notice that variable z appears in C, this is why assumption $\mathsf{w} : \mathsf{C}$ expresses that z satisfies C".

  This is the usual induction principle for the natural numbers, i.e. prove the base case (witnessed by g), and then prove the inductive step for the successor (witnessed by h).

  The $\mathbb{N}$-eliminator term $\mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}; \mathsf{n})$ is a *witness* for this proof by elimination. It remembers all the pieces: the predicate C, the proof g that C holds for 0, the proof h for the inductive step, and the arbitrary element n.

  The dot notation $\mathsf{z}.\mathsf{C}$ and $\mathsf{z}.\mathsf{w}.\mathsf{h}$ in term $\mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}; \mathsf{n})$ denotes bind structures (see Definition 1.5.51 below).

Convention 1.5.50 (Parentheses)**.** Given an S-term of the form:

$$\mathsf{ind}_{\mathbb{N}}[\mathsf{z.A}](\tau;\, \mathsf{y.w.}\nu;\, \mathsf{p})$$

we will omit the most external parentheses in S-terms A, τ, ν, and p. However, we may write them for emphasis or in case of ambiguity.

▲

We now introduce the bind structures for $\mathbb{N}$.

Definition 1.5.51 (Bind structures and local variables for $\mathbb{N}$)**.** In a $\mathbb{N}$-eliminator:

$$\mathsf{ind}_{\mathbb{N}}[\mathsf{z.A}](\nu;\, \mathsf{y.w.}\tau;\, \mathsf{p})$$

there are two *bind structures*:

- The expression $\mathsf{z.A}$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{z}\}$.
- The expression $\mathsf{y.w.}\tau$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{y}, \mathsf{w}\}$.

▲

It is time to state the computation rules. Since we have two constructors (i.e. two introduction rules), there will be two ways to give a constructor into an $\mathbb{N}$-eliminator. Hence, we will have two computation rules.

- Computation 1 (S-context: Γ; S-term: g, h, C; Variable: w, z):

$$\frac{\Gamma, \mathsf{z}\colon \mathbb{N} \vdash \mathsf{C}\colon \mathcal{U}_i \qquad \Gamma \vdash \mathsf{g}\colon \mathsf{C}[\mathsf{z}/\mathsf{0}] \qquad \Gamma, \mathsf{z}\colon \mathbb{N}, \mathsf{w}\colon \mathsf{C} \vdash \mathsf{h}\colon \mathsf{C}[\mathsf{z}/\mathsf{succ\ z}]}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}[\mathsf{z.C}](\mathsf{g};\, \mathsf{z.w.h};\, \mathsf{0}) \equiv \mathsf{g}\colon \mathsf{C}[\mathsf{z}/\mathsf{0}]}\ \mathbb{N}\text{-COMP-1}$$

  Informal reading. If the proof produced by the elimination rule is applied on $\mathsf{0}$, then this proof must be g, i.e. it must be the proof for the base case.

- Computation 2 (S-context: Γ; S-term: g, h, n, C; Variable: w, z):

$$\frac{\Gamma, \mathsf{z}\colon \mathbb{N} \vdash \mathsf{C}\colon \mathcal{U}_i \qquad \Gamma \vdash \mathsf{g}\colon \mathsf{C}[\mathsf{z}/\mathsf{0}] \qquad \Gamma, \mathsf{z}\colon \mathbb{N}, \mathsf{w}\colon \mathsf{C} \vdash \mathsf{h}\colon \mathsf{C}[\mathsf{z}/\mathsf{succ\ z}] \qquad \Gamma \vdash \mathsf{n}\colon \mathbb{N}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}[\mathsf{z.C}](\mathsf{g};\, \mathsf{z.w.h};\, \mathsf{succ\ n}) \equiv \mathsf{h}[\mathsf{z}/\mathsf{n}, \mathsf{w}/\mathsf{ind}_{\mathbb{N}}[\mathsf{z.C}](\mathsf{g};\, \mathsf{z.w.h};\, \mathsf{n})] : \mathsf{C}[\mathsf{z}/\mathsf{succ\ n}]}\ \mathbb{N}\text{-COMP-2}$$

  Informal reading. Suppose that the elimination rule was used to produce the proof $\mathsf{ind}_{\mathbb{N}}[\mathsf{z.C}](\mathsf{g};\, \mathsf{z.w.h};\, \mathsf{succ\ n})$, i.e. $(\mathsf{succ\ n})$ satisfies C. Then, the only way for this to be true is that the elimination rule was used to produce the proof $\mathsf{ind}_{\mathbb{N}}[\mathsf{z.C}](\mathsf{g};\, \mathsf{z.w.h};\, \mathsf{n})$ (i.e. n satisfies C), which then was passed into the proof for the inductive step h.

HoTT does not state a uniqueness rule for $\mathbb{N}$. Also, the elimination and computation rules can be encapsulated into an induction principle.

Lemma 1.5.52 (Induction principle for $\mathbb{N}$)**.** *Let* $\mathsf{C}\colon \mathbb{N} \to \mathcal{U}_i$ *be a type family. If we have:*

$$\mathsf{g}\colon \mathsf{C\ 0}$$
$$\mathsf{h}\colon \prod_{\mathsf{z}:\mathbb{N}} (\mathsf{C\ z}) \to (\mathsf{C\ (succ\ z)})$$

Then, we can define a function:

$$\mathsf{f}\colon \prod_{\mathsf{n}:\mathbb{N}} \mathsf{C\ n}$$

as:

$$\mathsf{f\ 0} :\equiv \mathsf{g}$$
$$\mathsf{f\ (succ\ m)} :\equiv \mathsf{h\ m\ (f\ m)}$$

where $\mathsf{m}\colon \mathbb{N}$.

Proof. We follow the same pattern.[20] First define:

$$\mathsf{f} :\equiv \lambda \mathsf{n} : \mathbb{N}.\ \mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}\ \mathsf{z}\ \mathsf{w}; \mathsf{n})$$

Now, we check that f has the required type. Suppose $\mathsf{n} : \mathbb{N}$. We need to check:

$$\mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}\ \mathsf{z}\ \mathsf{w}; \mathsf{n}) : \mathsf{C}\ \mathsf{n}$$

We will use the $\mathbb{N}$-ELIM rule, so we need to prove:

- Given $\mathsf{z} : \mathbb{N}$, term $\mathsf{C}\ \mathsf{z}$ is a type. Proof: apply Π-ELIM to get $\mathsf{C}\ \mathsf{z} : \mathcal{U}_i$.
- Check $\mathsf{g} : (\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{0}]$. Proof: by assumption, since $(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{0}]$ denotes $\mathsf{C}\ \mathsf{0}$.
- Given $\mathsf{z} : \mathbb{N}$ and $\mathsf{w} : \mathsf{C}\ \mathsf{z}$, check $\mathsf{h}\ \mathsf{z}\ \mathsf{w} : (\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{succ}\ \mathsf{z}]$. Proof: apply twice the type family version of Π-ELIM to get $\mathsf{h}\ \mathsf{z}\ \mathsf{w} : \mathsf{C}\ (\mathsf{succ}\ \mathsf{z})$. But $(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{succ}\ \mathsf{z}]$ denotes $\mathsf{C}\ (\mathsf{succ}\ \mathsf{z})$.
- Check $\mathsf{n} : \mathbb{N}$. Proof: by assumption.

Hence, we have:

$$\mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}\ \mathsf{z}\ \mathsf{w}; \mathsf{n}) : (\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{n}]$$

But $(\mathsf{C}\ \mathsf{z})[\mathsf{z}/\mathsf{n}]$ denotes $\mathsf{C}\ \mathsf{n}$.

Now, the first equation follows directly by definition and the $\mathbb{N}$-COMP-1 rule. For the second equation we have:

$$\begin{aligned}
\mathsf{f}\ (\mathsf{succ}\ \mathsf{m}) &\equiv (\lambda \mathsf{n}.\ \mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}\ \mathsf{z}\ \mathsf{w}; \mathsf{n}))\ (\mathsf{succ}\ \mathsf{m}) \\
&\equiv \mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}\ \mathsf{z}\ \mathsf{w}; \mathsf{succ}\ \mathsf{m}) \\
&\equiv (\mathsf{h}\ \mathsf{z}\ \mathsf{w})[\mathsf{z}/\mathsf{m}, \mathsf{w}/\mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}\ \mathsf{z}\ \mathsf{w}; \mathsf{m})] \\
&\equiv \mathsf{h}\ \mathsf{m}\ \mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}\ \mathsf{z}\ \mathsf{w}; \mathsf{m}) \\
&\equiv \mathsf{h}\ \mathsf{m}\ (\mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}\ \mathsf{z}\ \mathsf{w}; \mathsf{n})[\mathsf{n}/\mathsf{m}]) \\
&\equiv \mathsf{h}\ \mathsf{m}\ ((\lambda \mathsf{n}.\ \mathsf{ind}_{\mathbb{N}}[\mathsf{z}.\mathsf{C}\ \mathsf{z}](\mathsf{g}; \mathsf{z}.\mathsf{w}.\mathsf{h}\ \mathsf{z}\ \mathsf{w}; \mathsf{n}))\ \mathsf{m}) \\
&\equiv \mathsf{h}\ \mathsf{m}\ (\mathsf{f}\ \mathsf{m})
\end{aligned}$$

□

The induction principle reads (under the Curry-Howard correspondence): "Let C be a unary predicate on natural numbers. If we prove the base case $\mathsf{C}\ \mathsf{0}$ and the inductive step $(\mathsf{C}\ \mathsf{n}) \to (\mathsf{C}\ (\mathsf{succ}\ \mathsf{n}))$ for any natural n, then C will hold for all naturals".

But from the point of view of functions, the induction principle reads: "To define a function on natural numbers, simply define it by primitive recursion".

When family C in the induction principle is constant, we have a *recursion principle*.

Lemma 1.5.53 (Recursion principle for natural numbers)**.** *Let $\mathsf{D} : \mathcal{U}_i$ be a type. If we have:*

$$\mathsf{g} : \mathsf{D}$$
$$\mathsf{h} : \mathbb{N} \to \mathsf{D} \to \mathsf{D}$$

Then, we can define a function:

$$\mathsf{f} : \mathbb{N} \to \mathsf{D}$$

as:

$$\mathsf{f}\ \mathsf{0} :\equiv \mathsf{g}$$
$$\mathsf{f}\ (\mathsf{succ}\ \mathsf{m}) :\equiv \mathsf{h}\ \mathsf{m}\ (\mathsf{f}\ \mathsf{m})$$

where $\mathsf{m} : \mathbb{N}$.

Proof. Instantiate the induction principle with the constant family:

$$\mathsf{C} :\equiv (\lambda \mathsf{z}.\ \mathsf{D}) : \mathbb{N} \to \mathcal{U}_i$$

And simplify applications accordingly.

□

[20] We will do the details, as this is the first case in which we have a type with an inductive step on the elimination rule.

1.5.7 Identity types

Let A be a type, and α, β two terms of type A. Type $\alpha =_{\mathsf{A}} \beta$ will denote the *proposition* that α and β are *identifiable*. When we have a term $\tau : \alpha =_{\mathsf{A}} \beta$, this term τ will be a *proof* to the fact that α and β are identifiable. Hence, we call these types, *identity types*.

Do not confuse $=$ with definitional equality $\equiv$. Definitional equality is a judgment, while the identity type is, well, a type. It will be true that if α simplifies to β (i.e. judgment $\alpha \equiv \beta$ is derivable), then we will have that α and β are identifiable (i.e. $\alpha =_{\mathsf{A}} \beta$ is inhabited). But, the converse will not be true in general: if α and β are identifiable, then α does not necessarily simplify to β.

We provide an example to make the difference clear. Given a pair $\mathsf{p} : \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$, in Lemma 1.6.1 we will prove that the following type is inhabited:

$$(\mathsf{pr}_1\ \mathsf{p}, \mathsf{pr}_2\ \mathsf{p}) = \mathsf{p}$$

If we attempt to derive the judgment $(\mathsf{pr}_1\ \mathsf{p}, \mathsf{pr}_2\ \mathsf{p}) \equiv \mathsf{p}$ we will get stuck, because the projection functions do not know how to compute when a variable p is given as input to them, i.e. their defining equations only compute on canonical pairs (see Lemma 1.5.26). In other words, from the point of view of definitional equality, computing the first and second coordinates of a *variable itself* does not make any sense. Definitional equality is syntactical computation.

To the contrary, when we claim that $(\mathsf{pr}_1\ \mathsf{p}, \mathsf{pr}_2\ \mathsf{p}) = \mathsf{p}$ is inhabited, we are implicitly assuming that variable p stands for some arbitrary canonical pair (a, b). Hence, we are really claiming that type $(\mathsf{pr}_1\ (\mathsf{a}, \mathsf{b}), \mathsf{pr}_2\ (\mathsf{a}, \mathsf{b})) = (\mathsf{a}, \mathsf{b})$ is inhabited. In other words, type $(\mathsf{pr}_1\ \mathsf{p}, \mathsf{pr}_2\ \mathsf{p}) = \mathsf{p}$ is claiming a *general* property about pairs (i.e. it holds for any canonical pair).

We state the formation and introduction rules for identity types.

- Formation (S-context: Γ; S-term: α, β, A):

$$\frac{\Gamma \vdash \mathsf{A} : \mathcal{U}_i \qquad \Gamma \vdash \alpha : \mathsf{A} \qquad \Gamma \vdash \beta : \mathsf{A}}{\Gamma \vdash \alpha =_{\mathsf{A}} \beta : \mathcal{U}_i}\ =\text{-FORM}$$

  Informal reading. Let A be a type at level i. If α and β are terms of type A, then $\alpha =_{\mathsf{A}} \beta$ is a type at level i.

- Introduction (S-context: Γ; S-term: α, A, B):

$$\frac{\Gamma \vdash \mathsf{A} : \mathcal{U}_i \qquad \Gamma \vdash \alpha : \mathsf{A}}{\Gamma \vdash \mathsf{refl}_{\mathsf{A}}\ \alpha : \alpha =_{\mathsf{A}} \alpha}\ =\text{-INTRO}$$

  Informal reading. Let α be a term belonging to A. Then, term $\mathsf{refl}_{\mathsf{A}}\ \alpha$ is a proof expressing that α is identifiable with itself.

  Term $\mathsf{refl}_{\mathsf{A}}\ \alpha$ is called the *reflexivity proof* or the *trivial proof* for $\alpha =_{\mathsf{A}} \alpha$.

Convention 1.5.54. We follow the next conventions:

- In an S-term of the form $\alpha =_{\mathsf{A}} \beta$ we will omit the most external parentheses in S-terms α and β. However, sometimes we may write them for readability.
- Most of the time, in an S-term of the form $\alpha =_{\mathsf{A}} \beta$, we will omit the S-term A. For example, S-term $\mathbb{0} =_{\mathcal{U}_0} \mathbb{1}$ can be written as $\mathbb{0} = \mathbb{1}$.

▲

Now, we state the elimination rule for identity types.

- Elimination (S-context: Γ; S-term: α, β, g, q, A, C; Variable: w, y, z, p):

$$\frac{\begin{array}{c}\Gamma, \mathsf{w} : \mathsf{A},\ \mathsf{y} : \mathsf{A},\ \mathsf{p} : \mathsf{w} =_{\mathsf{A}} \mathsf{y} \vdash \mathsf{C} : \mathcal{U}_i \\ \Gamma, \mathsf{z} : \mathsf{A} \vdash \mathsf{g} : \mathsf{C}[\mathsf{w}/\mathsf{z}, \mathsf{y}/\mathsf{z}, \mathsf{p}/\mathsf{refl}_{\mathsf{A}}\ \mathsf{z}] \qquad \Gamma \vdash \alpha : \mathsf{A} \qquad \Gamma \vdash \beta : \mathsf{A} \qquad \Gamma \vdash \mathsf{q} : \alpha =_{\mathsf{A}} \beta\end{array}}{\Gamma \vdash \mathsf{ind}_{=}[\mathsf{w}.\mathsf{y}.\mathsf{p}.\mathsf{C}](\mathsf{z}.\mathsf{g}; \alpha; \beta; \mathsf{q}) : \mathsf{C}[\mathsf{w}/\alpha, \mathsf{y}/\beta, \mathsf{p}/\mathsf{q}]}\ =\text{-ELIM}$$

Informal reading. Condition $\Gamma, \mathsf{w}\colon \mathsf{A},\ \mathsf{y}\colon \mathsf{A},\ \mathsf{p}\colon \mathsf{w} =_{\mathsf{A}} \mathsf{y} \vdash \mathsf{C}\colon \mathcal{U}_i$ reads: "Let C be a predicate on identification proofs".

Conditions $\Gamma \vdash \alpha\colon \mathsf{A}$, $\Gamma \vdash \beta\colon \mathsf{A}$, and $\Gamma \vdash \mathsf{q}\colon \alpha =_{\mathsf{A}} \beta$ together read: "Let q be a proof that elements α and β are identifiable".

Then, the rest reads: "To prove that predicate C holds for q (i.e. type $\mathsf{C}[\mathsf{w}/\alpha, \mathsf{y}/\beta, \mathsf{p}/\mathsf{q}]$ in the conclusion), prove that predicate C holds for all reflexivity proofs on elements of A".

We are given that α and β are identifiable through a proof q. Since the introduction rule only constructs trivial proofs, term q should be decomposable into the form $\mathsf{refl}_{\mathsf{A}}\ \mathsf{b}$ for some $\mathsf{b}\colon \mathsf{A}$ (i.e. the only way to build q is through the introduction rule). Therefore, C should hold for q because C holds for all trivial proofs.

The explanation on the last paragraph seems natural enough. However, at some point in the future we will add inference rules that introduce identity proofs *different* from the reflexivity proof (see Sections 1.6.1 and 1.6.2). At the moment those rules are added, the explanation on the previous paragraph will start to seem doubtful, because reflexivity proofs will *not* be the only proofs for identities. So, we will have to revise what the elimination rule is *really* saying. The homotopy interpretation for HoTT, which will be presented in Section 1.7, will provide an explanation for why the elimination rule is still sound even though there are identity proofs different from the reflexivity proof.

The $=$-eliminator term $\mathsf{ind}_{=}[\mathsf{w.y.p.C}](\mathsf{z.g}; \alpha; \beta; \mathsf{q})$ is a *witness* for this proof by elimination. It remembers all the pieces: the predicate C, the proof g that C holds for reflexivity proofs, and the elements α, β with proof q, which witnesses their identification.

We follow the usual dot notation for describing bind structures in $\mathsf{ind}_{=}[\mathsf{w.y.p.C}](\mathsf{z.g}; \alpha; \beta; \mathsf{q})$.

Convention 1.5.55 (Parentheses)**.** Given an S-term of the form:

$$\mathsf{ind}_{=}[\mathsf{w.y.p.A}](\mathsf{z}.\tau; \alpha; \beta; \nu)$$

we will omit the most external parentheses in S-terms A, τ, α, β, and ν. However, we may write them for emphasis or in case of ambiguity.

▲

We now introduce the bind structures for identity types.

Definition 1.5.56 (Bind structures and local variables for identity types)**.** In a $=$-eliminator:

$$\mathsf{ind}_{=}[\mathsf{w.y.p.A}](\mathsf{z}.\tau; \alpha; \beta; \nu)$$

there are two *bind structures*:

- The expression $\mathsf{w.y.p.A}$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{w}, \mathsf{y}, \mathsf{p}\}$.
- The expression $\mathsf{z}.\tau$ is a *bind structure*. Its set of *local variables* is defined to be $\{\mathsf{z}\}$.

▲

It is time to state the computation rule.

- Computation (S-context: Γ; S-term: α, g, A, C; Variable: w, y, z, p):

$$\frac{\Gamma, \mathsf{w}\colon \mathsf{A},\ \mathsf{y}\colon \mathsf{A},\ \mathsf{p}\colon \mathsf{w} =_{\mathsf{A}} \mathsf{y} \vdash \mathsf{C}\colon \mathcal{U}_i \qquad \Gamma, \mathsf{z}\colon \mathsf{A} \vdash \mathsf{g}\colon \mathsf{C}[\mathsf{w}/\mathsf{z}, \mathsf{y}/\mathsf{z}, \mathsf{p}/\mathsf{refl}_{\mathsf{A}}\ \mathsf{z}] \qquad \Gamma \vdash \alpha\colon \mathsf{A}}{\Gamma \vdash \mathsf{ind}_{=}[\mathsf{w.y.p.C}](\mathsf{z.g}; \alpha; \alpha; \mathsf{refl}_{\mathsf{A}}\ \alpha) \equiv \mathsf{g}[\mathsf{z}/\alpha] : \mathsf{C}[\mathsf{w}/\alpha, \mathsf{y}/\alpha, \mathsf{p}/\mathsf{refl}_{\mathsf{A}}\ \alpha]}\ =\text{-COMP}$$

Informal reading. If we prove, using the *elimination* rule, that C holds for the trivial proof $\mathsf{refl}_{\mathsf{A}}\ \alpha$, then this proof must be g applied on α, because the elimination rule makes use of g to produce proofs for $\mathsf{C}[\mathsf{w}/\alpha, \mathsf{y}/\alpha, \mathsf{p}/\mathsf{refl}_{\mathsf{A}}\ \alpha]$.

HoTT does not state a uniqueness rule for identity types. Also, the elimination and computation rules can be encapsulated into an induction principle.

Lemma 1.5.57 (Induction principle for identity types)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *be a type.*
If we are given a family $\mathsf{C}:\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}}((\mathsf{w}=\mathsf{y})\to\mathcal{U}_j)$,[21] *and a function:*

$$\mathsf{g}:\prod_{\mathsf{z}:\mathsf{A}}\mathsf{C}\ \mathsf{z}\ \mathsf{z}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{z})$$

Then, we can define a function:

$$\mathsf{f}:\prod_{\mathsf{m},\mathsf{n}:\mathsf{A}}\ \prod_{\mathsf{q}:\mathsf{m}=\mathsf{n}}\mathsf{C}\ \mathsf{m}\ \mathsf{n}\ \mathsf{q}$$

as:

$$\mathsf{f}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) :\equiv \mathsf{g}\ \mathsf{a}$$

where $\mathsf{a}:\mathsf{A}$.

Proof. Define:

$$\mathsf{f} :\equiv \lambda(\mathsf{m}:\mathsf{A})(\mathsf{n}:\mathsf{A})(\mathsf{q}:\mathsf{m}=\mathsf{n}).\ \mathsf{ind}_{=}[\mathsf{w}.\mathsf{y}.\mathsf{p}.\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p}](\mathsf{z}.\mathsf{g}\ \mathsf{z};\ \mathsf{m};\ \mathsf{n};\ \mathsf{q})$$

Now, we check that f has the required type. Suppose $\mathsf{m}:\mathsf{A}$, $\mathsf{n}:\mathsf{A}$, and $\mathsf{q}:\mathsf{m}=\mathsf{n}$. We need to check:

$$\mathsf{ind}_{=}[\mathsf{w}.\mathsf{y}.\mathsf{p}.\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p}](\mathsf{z}.\mathsf{g}\ \mathsf{z};\ \mathsf{m};\ \mathsf{n};\ \mathsf{q}):\mathsf{C}\ \mathsf{m}\ \mathsf{n}\ \mathsf{q}$$

We will use the $=$-ELIM rule, so we need to prove:

- Given $\mathsf{w}:\mathsf{A}$, $\mathsf{y}:\mathsf{A}$, and $\mathsf{p}:\mathsf{w}=\mathsf{y}$, term $\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p}$ is a type. Proof: apply Π-ELIM three times to get $\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p}:\mathcal{U}_j$.
- Given $\mathsf{z}:\mathsf{A}$, check $\mathsf{g}\ \mathsf{z}:(\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p})[\mathsf{w}/\mathsf{z},\mathsf{y}/\mathsf{z},\mathsf{p}/\mathsf{refl}_{\mathsf{A}}\ \mathsf{z}]$. Proof: apply the type family version of Π-ELIM to get $\mathsf{g}\ \mathsf{z}:\mathsf{C}\ \mathsf{z}\ \mathsf{z}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{z})$.
  But $(\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p})[\mathsf{w}/\mathsf{z},\mathsf{y}/\mathsf{z},\mathsf{p}/\mathsf{refl}_{\mathsf{A}}\ \mathsf{z}]$ denotes $\mathsf{C}\ \mathsf{z}\ \mathsf{z}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{z})$.
- Check $\mathsf{m}:\mathsf{A}$. Proof: by assumption.
- Check $\mathsf{n}:\mathsf{A}$. Proof: by assumption.
- Check $\mathsf{q}:\mathsf{m}=\mathsf{n}$. Proof: by assumption.

Hence, we have:

$$\mathsf{ind}_{=}[\mathsf{w}.\mathsf{y}.\mathsf{p}.\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p}](\mathsf{z}.\mathsf{g}\ \mathsf{z};\ \mathsf{m};\ \mathsf{n};\ \mathsf{q}):(\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p})[\mathsf{w}/\mathsf{m},\mathsf{y}/\mathsf{n},\mathsf{p}/\mathsf{q}]$$

But $(\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p})[\mathsf{w}/\mathsf{m},\mathsf{y}/\mathsf{n},\mathsf{p}/\mathsf{q}]$ denotes $\mathsf{C}\ \mathsf{m}\ \mathsf{n}\ \mathsf{q}$.

Now, for the equation, we have by definition and computation:

$$\begin{aligned}\mathsf{f}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) &\equiv (\lambda\mathsf{m}.\ \lambda\mathsf{n}.\ \lambda\mathsf{q}.\ \mathsf{ind}_{=}[\mathsf{w}.\mathsf{y}.\mathsf{p}.\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p}](\mathsf{z}.\mathsf{g}\ \mathsf{z};\ \mathsf{m};\ \mathsf{n};\ \mathsf{q}))\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\\ &\equiv \mathsf{ind}_{=}[\mathsf{w}.\mathsf{y}.\mathsf{p}.\mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p}](\mathsf{z}.\mathsf{g}\ \mathsf{z};\ \mathsf{a};\ \mathsf{a};\ \mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\\ &\equiv (\mathsf{g}\ \mathsf{z})[\mathsf{z}/\mathsf{a}]\\ &\equiv \mathsf{g}\ \mathsf{a}\end{aligned}$$

□

The induction principle reads (under the Curry-Howard correspondence): "To prove that C holds for m and n when m and n are identifiable, it is enough to prove that C holds for trivial proofs".

But from the point of view of functions, the induction principle reads: "To define a function on identity types, just define it on trivial proofs $\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$ for arbitrary $\mathsf{a}:\mathsf{A}$, and freely use term $\mathsf{a}:\mathsf{A}$ inside the function body".

When family C in the induction principle is constant, we have a *recursion principle*.

[21] This function *is* a type family, but a family parameterized over three inputs instead of the usual form $\mathsf{E}\to\mathcal{U}_j$, which depends on one input.

It is possible to rewrite the induction principle so that it uses a standard family $\mathsf{C}:(\sum_{\mathsf{w},\mathsf{y}:\mathsf{A}}\mathsf{w}=\mathsf{y})\to\mathcal{U}_j$ (i.e. dependent on only one input), because types $(\sum_{\mathsf{w},\mathsf{y}:\mathsf{A}}\mathsf{w}=\mathsf{y})\to\mathcal{U}_j$ and $\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}}((\mathsf{w}=\mathsf{y})\to\mathcal{U}_j)$ are equivalent by Lemma 1.6.29. We will define the concept of type equivalence in Section 1.6.

Lemma 1.5.58 (Recursion principle for identity types)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *and* $\mathsf{D}:\mathcal{U}_j$ *be types. If we are given a function:*

$$\mathsf{g}:\mathsf{A}\to\mathsf{D}$$

Then, we can define a function:

$$\mathsf{f}:\prod_{\mathsf{m},\mathsf{n}:\mathsf{A}}((\mathsf{m}=\mathsf{n})\to\mathsf{D})$$

as:

$$\mathsf{f}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}):\equiv \mathsf{g}\ \mathsf{a}$$

where $\mathsf{a}:\mathsf{A}$.

Proof. Instantiate the induction principle with the constant family:

$$\mathsf{C}:\equiv(\lambda\mathsf{w}.\ \lambda\mathsf{y}.\ \lambda\mathsf{p}.\ \mathsf{D}):\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}}((\mathsf{w}=\mathsf{y})\to\mathcal{U}_j)$$

And simplify applications accordingly.

□

To show how the induction principle is applied, we define some functions that will be prominently used on this report.

The first one is the *symmetry operator*. It takes a proof $\mathsf{q}:\mathsf{a}=\mathsf{b}$ and returns a proof for the symmetric statement $(\mathsf{b}=\mathsf{a})$. Also, the operator acts as proof for the statement that the identity type is symmetric.

Lemma 1.5.59 (Symmetry operator)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *be a type. We define a function:*[22]

$$^{-1}:\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}}(\mathsf{w}=\mathsf{y})\to(\mathsf{y}=\mathsf{w})$$

called the symmetry operator*, as:*

$$^{-1}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}):\equiv \mathsf{refl}_{\mathsf{A}}\ \mathsf{a} \tag{1.8}$$

where $\mathsf{a}:\mathsf{A}$.

Given $\mathsf{w}:\mathsf{A}$, $\mathsf{y}:\mathsf{A}$, *and* $\mathsf{p}:\mathsf{w}=\mathsf{y}$, *instead of* $(^{-1}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p})$ *we will write* p^{-1}, *where inputs* w *and* y *are left implicit. Hence, Equation* (1.8) *can be written as:*

$$(\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})^{-1}:\equiv \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$$

Proof. By the induction principle instantiated with the family:[23]

$$\mathsf{C}:\equiv(\lambda\mathsf{w}.\ \lambda\mathsf{y}.\ \lambda\mathsf{p}.\ \mathsf{y}=\mathsf{w}):\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}}((\mathsf{w}=\mathsf{y})\to\mathcal{U}_i)$$

if we can construct a function:

$$\mathsf{g}:\prod_{\mathsf{z}:\mathsf{A}}\mathsf{C}\ \mathsf{z}\ \mathsf{z}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{z})$$

or equivalently (after simplification):

$$\mathsf{g}:\prod_{\mathsf{z}:\mathsf{A}}\mathsf{z}=\mathsf{z}$$

then, there will be a function:

$$^{-1}:\prod_{\mathsf{m},\mathsf{n}:\mathsf{A}}\prod_{\mathsf{q}:\mathsf{m}=\mathsf{n}}\mathsf{C}\ \mathsf{m}\ \mathsf{n}\ \mathsf{q}$$

[22] The symbol $^{-1}$ is the name of the function.

[23] Since A is at level i, type $\mathsf{y}=\mathsf{w}$ inside the lambda is at level i. See the formation rule for the identity type on page 45.

or equivalently (after simplification):

$$^{-1} : \prod_{m,n:A} \prod_{q:m=n} n = m$$

which can be written as (since q does not occur in $n = m$):

$$^{-1} : \prod_{m,n:A} (m = n) \to (n = m)$$

So, define:

$$g :\equiv (\lambda z.\ \mathsf{refl}_A\ z) : \prod_{z:A} z = z$$

And we have:

$$^{-1}\ a\ a\ (\mathsf{refl}_A\ a) \equiv g\ a \equiv (\lambda z.\ \mathsf{refl}_A\ z)\ a \equiv \mathsf{refl}_A\ a$$

□

The next function is the *transitivity operator*. It takes two proofs $p : a = b$, $q : b = c$ and returns a proof for the transitive statement $(a = c)$. Also, the operator acts as proof for the statement that the identity type is transitive.

Lemma 1.5.60 (Transitivity operator)**.** *Let $A : \mathcal{U}_i$ be a type. We define a function:*[24]

$$\centerdot : \prod_{w,y,z:A} (w = y) \to (y = z) \to (w = z)$$

called the transitivity operator*, as:*

$$\centerdot\ a\ a\ a\ (\mathsf{refl}_A\ a)\ (\mathsf{refl}_A\ a) :\equiv \mathsf{refl}_A\ a \tag{1.9}$$

where $a : A$.

Given $w : A$, $y : A$, $z : A$, $p : w = y$, and $q : y = z$, instead of $\centerdot\ w\ y\ z\ p\ q$, we will write $p \centerdot q$, where inputs w, y, and z are left implicit.

Hence, Equation (1.9) can be written as:

$$(\mathsf{refl}_A\ a) \centerdot (\mathsf{refl}_A\ a) :\equiv \mathsf{refl}_A\ a$$

Proof. If we can construct a function:

$$f : \prod_{w,y:A} (w = y) \to \left(\prod_{z:A} (y = z) \to (w = z) \right)$$

then, we will be able to define the transitivity operator as:

$$\centerdot :\equiv (\lambda w.\ \lambda y.\ \lambda z.\ \lambda p.\ \lambda q.\ f\ w\ y\ p\ z\ q) : \prod_{w,y,z:A} (w = y) \to (y = z) \to (w = z)$$

So, we focus on constructing function f. To this end, we apply the induction principle for identity types with the family:[25]

$$C :\equiv \left(\lambda w.\ \lambda y.\ \lambda p. \prod_{z:A} (y = z) \to (w = z) \right) : \prod_{w,y:A} ((w = y) \to \mathcal{U}_i)$$

So, f will exist if we are able to construct a function:

$$g : \prod_{t:A} C\ t\ t\ (\mathsf{refl}_A\ t)$$

[24] The symbol $\centerdot$ is the name of the function.

[25] Type A is at level i, therefore, by the $=$-FORM rule, types $y = z$ and $w = z$ are at level i. So, by Remark 1.5.16, type $\prod_{z:A} (y = z) \to (w = z)$ is at level $\max\{i, i, i\} = i$.

or equivalently (after simplification):

$$\mathsf{g}\colon \prod_{\mathsf{t}:\mathsf{A}} \prod_{\mathsf{z}:\mathsf{A}} (\mathsf{t} = \mathsf{z}) \to (\mathsf{t} = \mathsf{z})$$

But we will construct this function g by induction again, but this time with the family:

$$\mathsf{E} :\equiv (\lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \lambda \mathsf{p}.\ \mathsf{w} = \mathsf{y})\colon \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} ((\mathsf{w} = \mathsf{y}) \to \mathcal{U}_i)$$

So, g will exist if we are able to construct a function:

$$\mathsf{h}\colon \prod_{\mathsf{r}:\mathsf{A}} \mathsf{E}\ \mathsf{r}\ \mathsf{r}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{r})$$

or equivalently (after simplification):

$$\mathsf{h}\colon \prod_{\mathsf{r}:\mathsf{A}} \mathsf{r} = \mathsf{r}$$

So, define:

$$\mathsf{h} :\equiv (\lambda \mathsf{r}.\ \mathsf{refl}_{\mathsf{A}}\ \mathsf{r})\colon \prod_{\mathsf{r}:\mathsf{A}} \mathsf{r} = \mathsf{r}$$

Then, the induction principle tells us that g satisfies the following equation (for arbitrary input $\mathsf{a}\colon \mathsf{A}$):

$$\mathsf{g}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{h}\ \mathsf{a} \equiv \mathsf{refl}_{\mathsf{A}}\ \mathsf{a} \tag{1.10}$$

But now that g is defined, we satisfied the condition required for the existence of function f. Hence, f exists, and the induction principle tells us that f satisfies the following equation (for arbitrary input $\mathsf{a}\colon \mathsf{A}$):

$$\mathsf{f}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{g}\ \mathsf{a} \tag{1.11}$$

Hence, we have by definition and computation rules:

$$\begin{aligned}
\mathsf{a}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) &\equiv \mathsf{f}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \\
&\equiv (\mathsf{f}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}))\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \\
&\overset{(1)}{\equiv} (\mathsf{g}\ \mathsf{a})\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \\
&\equiv \mathsf{g}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \\
&\overset{(2)}{\equiv} \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}
\end{aligned}$$

where (1) follows by (1.11), and (2) by (1.10).

□

The next function is the *transport operator*. This operator acts as proof for the statement that identical terms satisfy the same properties.

Lemma 1.5.61 (Transport operator)**.** *Let* $\mathsf{A}\colon \mathcal{U}_i$ *be a type. We define a function:*

$$\mathsf{transport}\colon \prod_{\mathsf{P}:\mathsf{A}\to\mathcal{U}_j} \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} (\mathsf{w} = \mathsf{y}) \to (\mathsf{P}\ \mathsf{w}) \to (\mathsf{P}\ \mathsf{y})$$

called the transport operator*, as:*[26]

$$\mathsf{transport}\ \mathsf{Q}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) :\equiv \mathsf{id}_{(\mathsf{Q}\ \mathsf{a})} \tag{1.12}$$

where $\mathsf{Q}\colon \mathsf{A} \to \mathcal{U}_j$ *and* $\mathsf{a}\colon \mathsf{A}$.

[26] Here, $\mathsf{id}_{(\mathsf{Q}\ \mathsf{a})}$ is the identity function on $\mathsf{Q}\ \mathsf{a}$, see Definition 1.5.6.

Given $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$, $\mathsf{w} : \mathsf{A}$, $\mathsf{y} : \mathsf{A}$, *and* $\mathsf{q} : \mathsf{w} = \mathsf{y}$, *instead of* $\mathsf{transport}\ \mathsf{P}\ \mathsf{w}\ \mathsf{y}\ \mathsf{q}$ *we will write* $\mathsf{transport}^{\mathsf{P}}\ \mathsf{q}$, *where inputs* w *and* y *are left implicit. Also, when parameter* P *is clear from context, we write* q_* *instead of* $\mathsf{transport}^{\mathsf{P}}\ \mathsf{q}$.

Hence, Equation (1.12) *can be written as:*

$$\mathsf{transport}^{\mathsf{Q}}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) :\equiv \mathsf{id}_{(\mathsf{Q}\ \mathsf{a})}$$

or:

$$(\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})_* :\equiv \mathsf{id}_{(\mathsf{Q}\ \mathsf{a})}$$

Proof. Define:

$$\mathsf{transport} :\equiv \lambda \mathsf{P} : \mathsf{A} \to \mathcal{U}_j.\ \mathsf{f}_{\mathsf{P}}$$

where:

$$\mathsf{f}_{\mathsf{P}} : \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} (\mathsf{w} = \mathsf{y}) \to (\mathsf{P}\ \mathsf{w}) \to (\mathsf{P}\ \mathsf{y})$$

is a function we will define by induction on identity types (here, the notation f_{P} emphasizes that term f is dependent on variable P).

By using the family:

$$\mathsf{C}_{\mathsf{P}} :\equiv (\lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \lambda \mathsf{p}.\ (\mathsf{P}\ \mathsf{w}) \to (\mathsf{P}\ \mathsf{y})) : \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} ((\mathsf{w} = \mathsf{y}) \to \mathcal{U}_j)$$

the induction principle says that f_{P} will exist if we can construct a function:

$$\mathsf{g}_{\mathsf{P}} : \prod_{\mathsf{z}:\mathsf{A}} \mathsf{C}_{\mathsf{P}}\ \mathsf{z}\ \mathsf{z}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{z})$$

or equivalently (after simplification):

$$\mathsf{g}_{\mathsf{P}} : \prod_{\mathsf{z}:\mathsf{A}} (\mathsf{P}\ \mathsf{z}) \to (\mathsf{P}\ \mathsf{z})$$

So, define:

$$\mathsf{g}_{\mathsf{P}} :\equiv \lambda \mathsf{z} : \mathsf{A}.\ \mathsf{id}_{(\mathsf{P}\ \mathsf{z})}$$

where $\mathsf{id}_{(\mathsf{P}\ \mathsf{z})}$ is the identity function on $\mathsf{P}\ \mathsf{z}$.

Therefore, f_{P} exists and satisfies the equation:

$$\mathsf{f}_{\mathsf{P}}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{g}_{\mathsf{P}}\ \mathsf{a} \equiv \mathsf{id}_{(\mathsf{P}\ \mathsf{a})}$$

Hence, we finally have:

$$\begin{aligned} \mathsf{transport}\ \mathsf{Q}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) &\equiv (\lambda \mathsf{P}.\ \mathsf{f}_{\mathsf{P}})\ \mathsf{Q}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \\ &\equiv \mathsf{f}_{\mathsf{Q}}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \\ &\equiv \mathsf{id}_{(\mathsf{Q}\ \mathsf{a})} \end{aligned}$$

□

The next function is the *application operator*. This operator acts as proof for the statement that identical inputs to a function produce identical outputs.

Lemma 1.5.62 (Application operator)**.** *Let* $\mathsf{A} : \mathcal{U}_i$ *and* $\mathsf{B} : \mathcal{U}_j$ *be types. We define a function:*

$$\mathsf{ap} : \prod_{\mathsf{f}:\mathsf{A}\to\mathsf{B}} \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} (\mathsf{w} = \mathsf{y}) \to (\mathsf{f}\ \mathsf{w} = \mathsf{f}\ \mathsf{y})$$

called the application operator*, as:*

$$\mathsf{ap}\ \mathsf{h}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) :\equiv \mathsf{refl}_{\mathsf{B}}\ (\mathsf{h}\ \mathsf{a}) \tag{1.13}$$

where $\mathsf{h} : \mathsf{A} \to \mathsf{B}$ *and* $\mathsf{a} : \mathsf{A}$.

Given $\mathsf{f} : \mathsf{A} \to \mathsf{B}$, $\mathsf{w} : \mathsf{A}$, $\mathsf{y} : \mathsf{A}$, *and* $\mathsf{q} : \mathsf{w} = \mathsf{y}$, *instead of* $\mathsf{ap}\ \mathsf{f}\ \mathsf{w}\ \mathsf{y}\ \mathsf{q}$ *we will write* $\mathsf{ap}_{\mathsf{f}}\ \mathsf{q}$, *where inputs* w *and* y *are left implicit.*

Hence, Equation (1.13) *can be written as:*

$$\mathsf{ap}_{\mathsf{h}}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) :\equiv \mathsf{refl}_{\mathsf{B}}\ (\mathsf{h}\ \mathsf{a})$$

Proof. Define:

$$\mathsf{ap} :\equiv \lambda \mathsf{f}:\mathsf{A} \to \mathsf{B}.\ \mathsf{k_f}$$

where:

$$\mathsf{k_f}: \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} (\mathsf{w} = \mathsf{y}) \to (\mathsf{f}\ \mathsf{w} = \mathsf{f}\ \mathsf{y})$$

is a function we will define by induction on identity types.

By using the family:

$$\mathsf{C_f} :\equiv (\lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \lambda \mathsf{p}.\ \mathsf{f}\ \mathsf{w} = \mathsf{f}\ \mathsf{y}): \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} ((\mathsf{w} = \mathsf{y}) \to \mathcal{U}_j)$$

the induction principle says that $\mathsf{k_f}$ will exist if we can construct a function:

$$\mathsf{g_f}: \prod_{\mathsf{z}:\mathsf{A}} \mathsf{C_f}\ \mathsf{z}\ \mathsf{z}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{z})$$

or equivalently (after simplification):

$$\mathsf{g_f}: \prod_{\mathsf{z}:\mathsf{A}} \mathsf{f}\ \mathsf{z} = \mathsf{f}\ \mathsf{z}$$

So, define:

$$\mathsf{g_f} :\equiv \lambda \mathsf{z}:\mathsf{A}.\ \mathsf{refl}_{\mathsf{B}}\ (\mathsf{f}\ \mathsf{z})$$

Therefore, $\mathsf{k_f}$ exists and satisfies the equation:

$$\mathsf{k_f}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{g_f}\ \mathsf{a} \equiv \mathsf{refl}_{\mathsf{B}}\ (\mathsf{f}\ \mathsf{a})$$

Hence, we finally have:

$$\begin{aligned} \mathsf{ap}\ \mathsf{h}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) &\equiv (\lambda \mathsf{f}.\ \mathsf{k_f})\ \mathsf{h}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \\ &\equiv \mathsf{k_h}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \\ &\equiv \mathsf{refl}_{\mathsf{B}}\ (\mathsf{h}\ \mathsf{a}) \end{aligned}$$

□

We also have a version for the application operator when the involved function is dependent.

Lemma 1.5.63 (Dependent application operator)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *be a type and* $\mathsf{P}:\mathsf{A}\to\mathcal{U}_j$ *a type family. We define a function:*

$$\mathsf{apd}: \prod_{\mathsf{f}:\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} \prod_{\mathsf{q}:\mathsf{w}=\mathsf{y}} \mathsf{transport}^{\mathsf{P}}\ \mathsf{q}\ (\mathsf{f}\ \mathsf{w}) = \mathsf{f}\ \mathsf{y}$$

called the dependent application operator*, as:*[27]

$$\mathsf{apd}\ \mathsf{h}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) :\equiv \mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ (\mathsf{h}\ \mathsf{a}) \tag{1.14}$$

where $\mathsf{h}: \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ *and* $\mathsf{a}:\mathsf{A}$.

Given $\mathsf{f}: \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$, $\mathsf{w}:\mathsf{A}$, $\mathsf{y}:\mathsf{A}$, *and* $\mathsf{q}:\mathsf{w}=\mathsf{y}$, *instead of* $\mathsf{apd}\ \mathsf{f}\ \mathsf{w}\ \mathsf{y}\ \mathsf{q}$ *we will write* $\mathsf{apd_f}\ \mathsf{q}$, *where inputs* w *and* y *are left implicit.*

Hence, Equation (1.14) *can be written as:*

$$\mathsf{apd_h}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) :\equiv \mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ (\mathsf{h}\ \mathsf{a})$$

[27] Notice this equation is well-typed, because $\mathsf{transport}^{\mathsf{P}}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\ (\mathsf{h}\ \mathsf{a})$ simplifies to $(\mathsf{h}\ \mathsf{a})$ by Lemma 1.5.61.

Proof. Define:

$$\mathsf{apd} :\equiv \lambda \mathsf{f} : \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}.\ \mathsf{k_f}$$

where:

$$\mathsf{k_f} : \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} \prod_{\mathsf{q}:\mathsf{w}=\mathsf{y}} \mathsf{transport}^{\mathsf{P}}\ \mathsf{q}\ (\mathsf{f}\ \mathsf{w}) = \mathsf{f}\ \mathsf{y}$$

is a function we will define by induction on identity types.

By using the family:

$$\mathsf{C_f} :\equiv (\lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \lambda \mathsf{q}.\ \mathsf{transport}^{\mathsf{P}}\ \mathsf{q}\ (\mathsf{f}\ \mathsf{w}) = \mathsf{f}\ \mathsf{y}) : \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} ((\mathsf{w} = \mathsf{y}) \to \mathcal{U}_j)$$

the induction principle says that $\mathsf{k_f}$ will exist if we can construct a function:

$$\mathsf{g_f} : \prod_{\mathsf{z}:\mathsf{A}} \mathsf{C_f}\ \mathsf{z}\ \mathsf{z}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{z})$$

or equivalently, by definition of $\mathsf{C_f}$:

$$\mathsf{g_f} : \prod_{\mathsf{z}:\mathsf{A}} \mathsf{transport}^{\mathsf{P}}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{z})\ (\mathsf{f}\ \mathsf{z}) = \mathsf{f}\ \mathsf{z}$$

or equivalently, after simplification (because $\mathsf{transport}^{\mathsf{P}}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{z}) \equiv \mathsf{id}_{(\mathsf{P}\ \mathsf{z})}$ by Lemma 1.5.61):

$$\mathsf{g_f} : \prod_{\mathsf{z}:\mathsf{A}} \mathsf{f}\ \mathsf{z} = \mathsf{f}\ \mathsf{z}$$

So, define:

$$\mathsf{g_f} :\equiv \lambda \mathsf{z} : \mathsf{A}.\ \mathsf{refl}_{(\mathsf{P}\ \mathsf{z})}\ (\mathsf{f}\ \mathsf{z})$$

Therefore, $\mathsf{k_f}$ exists and satisfies the equation:

$$\mathsf{k_f}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{g_f}\ \mathsf{a} \equiv \mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ (\mathsf{f}\ \mathsf{a})$$

Hence, we finally have:

$$\begin{aligned} \mathsf{apd}\ \mathsf{h}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) &\equiv (\lambda \mathsf{f}.\ \mathsf{k_f})\ \mathsf{h}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \\ &\equiv \mathsf{k_h}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \\ &\equiv \mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ (\mathsf{h}\ \mathsf{a}) \end{aligned}$$

□

1.6 Basic properties and equivalences

The first properties are propositional uniqueness principles. For Σ-types, their uniqueness principle will express that any pair is identifiable with a canonical pair.

Lemma 1.6.1 (Uniqueness principle for Σ-types)**.** *Let* $\mathsf{A} : \mathcal{U}_i$ *be a type and* $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$ *a family. Then, the following type is inhabited:*

$$\prod_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} (\mathsf{pr}_1\ \mathsf{q}, \mathsf{pr}_2\ \mathsf{q}) = \mathsf{q}$$

Proof. By the induction principle for Σ-types (Lemma 1.5.24) applied with the family:[28]

$$\mathsf{C} :\equiv (\lambda \mathsf{t}.\ (\mathsf{pr}_1\ \mathsf{t}, \mathsf{pr}_2\ \mathsf{t}) = \mathsf{t}) : \left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right) \to \mathcal{U}_{\max\{i,j\}}$$

[28] See remark 1.5.29 on universe level for Σ-types. See also the formation rule for identity types on page 45.

it is enough to prove that the following type is inhabited:

$$\prod_{a:A} \prod_{b:P\ a} C\ (a, b)$$

or equivalently:

$$\prod_{a:A} \prod_{b:P\ a} (\mathsf{pr}_1\ (a, b), \mathsf{pr}_2\ (a, b)) = (a, b)$$

or equivalently (after projections are simplified):

$$\prod_{a:A} \prod_{b:P\ a} (a, b) = (a, b)$$

But this type is inhabited by the function:

$$\lambda(a:A)(b:P\ a).\ \mathsf{refl}_{\left(\sum_{w:A} P\ w\right)}\ (a, b)$$

□

The uniqueness principle for $\mathbb{1}$ expresses that any term in $\mathbb{1}$ is identifiable with $\star$.

Lemma 1.6.2 (Uniqueness principle for the unit type)**.** *The following type is inhabited:*

$$\prod_{w:\mathbb{1}} w = \star$$

Proof. By the induction principle for $\mathbb{1}$ (Lemma 1.5.47), it is enough to prove that the following type is inhabited:[29]

$$\star = \star$$

But this type is inhabited by $(\mathsf{refl}_{\mathbb{1}}\ \star)$.

□

We also have some properties regarding function composition.

Lemma 1.6.3 (Associativity of function composition)**.** *Let* $A:\mathcal{U}_i$, $B:\mathcal{U}_j$, $C:\mathcal{U}_k$, $D:\mathcal{U}_l$ *be types, and* $f:A \to B$, $g:B \to C$, $h:C \to D$ *three functions. Then, the following judgment is derivable:*[30]

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

Proof. By definition:

$$\begin{aligned}(h \circ g) \circ f &\equiv \lambda t.\ (h \circ g)\ (f\ t)\\ &\equiv \lambda t.\ (\lambda w.\ h\ (g\ w))\ (f\ t)\\ &\equiv \lambda t.\ h\ (g\ (f\ t))\end{aligned}$$

And:

$$\begin{aligned}h \circ (g \circ f) &\equiv \lambda t.\ h\ ((g \circ f)\ t)\\ &\equiv \lambda t.\ h\ ((\lambda w.\ g\ (f\ w))\ t)\\ &\equiv \lambda t.\ h\ (g\ (f\ t))\end{aligned}$$

□

[29] We are using the family:

$$C :\equiv (\lambda w.\ w = \star) : \mathbb{1} \to \mathcal{U}_i$$

From now on, the involved family will not be explicitly stated, unless it is not obvious.

[30] Here, $\circ$ is function composition, see Definition 1.5.13.

Lemma 1.6.4 (Composition of identities)**.** *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$ *be types, and* $\mathsf{f}:\mathsf{A}\to\mathsf{B}$ *a function. Then, the following two judgments are derivable:*

$$\mathsf{id}_{\mathsf{B}}\circ\mathsf{f}\equiv\mathsf{f}$$
$$\mathsf{f}\circ\mathsf{id}_{\mathsf{A}}\equiv\mathsf{f}$$

Proof. By definition:

$$\begin{aligned}\mathsf{id}_{\mathsf{B}}\circ\mathsf{f}&\equiv\lambda\mathsf{t}.\ \mathsf{id}_{\mathsf{B}}\ (\mathsf{f}\ \mathsf{t})\\&\equiv\lambda\mathsf{t}.\ \mathsf{f}\ \mathsf{t}\\&\equiv\mathsf{f}\end{aligned}$$

where the last step follows by the uniqueness rule for Π-types on page 21.

For the other judgment:

$$\begin{aligned}\mathsf{f}\circ\mathsf{id}_{\mathsf{A}}&\equiv\lambda\mathsf{t}.\ \mathsf{f}\ (\mathsf{id}_{\mathsf{A}}\ \mathsf{t})\\&\equiv\lambda\mathsf{t}.\ \mathsf{f}\ \mathsf{t}\\&\equiv\mathsf{f}\end{aligned}$$

□

Proofs for identity types have the following properties.

Lemma 1.6.5. *Let* $\mathsf{A}:\mathcal{U}_i$*. The following types are inhabited:*[31]

(i)
$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}}\ \prod_{\mathsf{p}:\mathsf{w}=\mathsf{y}}\ \mathsf{p}=\mathsf{p}\cdot(\mathsf{refl}_{\mathsf{A}}\ \mathsf{y})$$

(ii)
$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}}\ \prod_{\mathsf{p}:\mathsf{w}=\mathsf{y}}\ \mathsf{p}=(\mathsf{refl}_{\mathsf{A}}\ \mathsf{w})\cdot\mathsf{p}$$

(iii)
$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}}\ \prod_{\mathsf{p}:\mathsf{w}=\mathsf{y}}\ \mathsf{p}^{-1}\cdot\mathsf{p}=\mathsf{refl}_{\mathsf{A}}\ \mathsf{y}$$

(iv)
$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}}\ \prod_{\mathsf{p}:\mathsf{w}=\mathsf{y}}\ \mathsf{p}\cdot\mathsf{p}^{-1}=\mathsf{refl}_{\mathsf{A}}\ \mathsf{w}$$

(v)
$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}}\ \prod_{\mathsf{p}:\mathsf{w}=\mathsf{y}}\ (\mathsf{p}^{-1})^{-1}=\mathsf{p}$$

Proof. We prove each case as follows.

(i). By the induction principle for identity types, it is enough to prove:

$$\prod_{\mathsf{a}:\mathsf{A}}\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}=(\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\cdot(\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})$$

or equivalently:

$$\prod_{\mathsf{a}:\mathsf{A}}\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}=\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$$

[31] Here, $\cdot$ is the transitivity operator, see Lemma 1.5.60. Also, $^{-1}$ is the symmetry operator, see Lemma 1.5.59.

since $(\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \cdot (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$ by Lemma 1.5.60.
So, let $\mathsf{a}\colon \mathsf{A}$. We have $\mathsf{refl}_{(\mathsf{a}=\mathsf{a})}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\colon \mathsf{refl}_{\mathsf{A}}\ \mathsf{a} = \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$.

(ii). The argument is similar to (i).

(iii). By the induction principle for identity types, it is enough to prove:

$$\prod_{\mathsf{a}:\mathsf{A}} (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})^{-1} \cdot (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) = \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$$

or equivalently:

$$\prod_{\mathsf{a}:\mathsf{A}} \mathsf{refl}_{\mathsf{A}}\ \mathsf{a} = \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$$

since $(\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})^{-1} \equiv \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$ by Lemma 1.5.59, and $(\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \cdot (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$ by Lemma 1.5.60.
So, let $\mathsf{a}\colon \mathsf{A}$. We have $\mathsf{refl}_{(\mathsf{a}=\mathsf{a})}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\colon \mathsf{refl}_{\mathsf{A}}\ \mathsf{a} = \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$.

(iv). The argument is similar to (iii).

(v). By the induction principle for identity types, it is enough to prove:

$$\prod_{\mathsf{a}:\mathsf{A}} ((\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})^{-1})^{-1} = \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$$

or equivalently:

$$\prod_{\mathsf{a}:\mathsf{A}} \mathsf{refl}_{\mathsf{A}}\ \mathsf{a} = \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$$

since $(\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})^{-1} \equiv \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$ by Lemma 1.5.59.
So, let $\mathsf{a}\colon \mathsf{A}$. We have $\mathsf{refl}_{(\mathsf{a}=\mathsf{a})}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\colon \mathsf{refl}_{\mathsf{A}}\ \mathsf{a} = \mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$.

□

The next lemma expresses that transporting on a constant family has no effects.

Lemma 1.6.6. *Let $\mathsf{A}\colon \mathcal{U}_i$ and $\mathsf{B}\colon \mathcal{U}_j$ be two types. Then, the following type is inhabited.*[32]

$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} \prod_{\mathsf{q}:\mathsf{w}=\mathsf{y}} \prod_{\mathsf{b}:\mathsf{B}} \mathsf{transport}^{(\lambda \mathsf{r}.\ \mathsf{B})}\ \mathsf{q}\ \mathsf{b} = \mathsf{b}$$

Proof. By the induction principle for identity types, it is enough to prove:

$$\prod_{\mathsf{a}:\mathsf{A}} \prod_{\mathsf{b}:\mathsf{B}} \mathsf{transport}^{(\lambda \mathsf{r}.\ \mathsf{B})}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\ \mathsf{b} = \mathsf{b}$$

or equivalently:

$$\prod_{\mathsf{a}:\mathsf{A}} \prod_{\mathsf{b}:\mathsf{B}} \mathsf{b} = \mathsf{b}$$

since $\mathsf{transport}^{(\lambda \mathsf{r}.\ \mathsf{B})}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{id}_{((\lambda \mathsf{r}.\ \mathsf{B})\ \mathsf{a})} \equiv \mathsf{id}_{\mathsf{B}}$ by Lemma 1.5.61.
Therefore, $(\lambda \mathsf{a}.\ \lambda \mathsf{b}.\ \mathsf{refl}_{\mathsf{B}}\ \mathsf{b})$ inhabits our type.

□

In order to define what an equivalence is, we need to introduce the concept of pointwise identifiable functions.

Definition 1.6.7 (Pointwise identifiable functions)**.** Let $\mathsf{A}\colon \mathcal{U}_i$ be a type, $\mathsf{P}\colon \mathsf{A} \to \mathcal{U}_j$ a family, and $\mathsf{f}, \mathsf{g}\colon \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ two functions. We define type:

$$\mathsf{f} \sim \mathsf{g} :\equiv \prod_{\mathsf{w}:\mathsf{A}} \mathsf{f}\ \mathsf{w} = \mathsf{g}\ \mathsf{w}$$

When type $\mathsf{f} \sim \mathsf{g}$ is inhabited, we say that f and g are *pointwise identifiable*.

▲

[32] The transport operator is defined in Lemma 1.5.61.

Under the Curry-Howard correspondence, type $\mathsf{f} \sim \mathsf{g}$ expresses: "For any $\mathsf{w}\colon \mathsf{A}$, functions f and g have identifiable outputs when given w as input".

In standard mathematics, if we know that two functions are pointwise equal, then we can conclude that both functions are *equal*. However, it is not possible to prove in our type theory *so far* that $(\mathsf{f} = \mathsf{g})$ is inhabited whenever f and g are pointwise identifiable. To fix this problem and force our functions to behave as in standard mathematics, we will introduce the function extensionality axiom in the next subsection.

We now show a couple of results involving pointwise identifiability. The first one expresses that type $\mathsf{f} \sim \mathsf{g}$ is an equivalence relation on functions.

Lemma 1.6.8. *Let $\mathsf{A}\colon \mathcal{U}_i$ be a type and $\mathsf{P}\colon \mathsf{A} \to \mathcal{U}_j$ a type family. Then, the following types are inhabited:*

(i) Reflexivity:

$$\prod_{\mathsf{f}:\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\,\mathsf{w}} \mathsf{f} \sim \mathsf{f}$$

(ii) Symmetry:

$$\prod_{\mathsf{f},\mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\,\mathsf{w}} (\mathsf{f} \sim \mathsf{g}) \to (\mathsf{g} \sim \mathsf{f})$$

(iii) Transitivity:

$$\prod_{\mathsf{f},\mathsf{g},\mathsf{h}:\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\,\mathsf{w}} (\mathsf{f} \sim \mathsf{g}) \to (\mathsf{g} \sim \mathsf{h}) \to (\mathsf{f} \sim \mathsf{h})$$

Proof. We prove each case as follows.

(i). Let $\mathsf{f}\colon \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\,\mathsf{w}$. Then:

$$(\lambda \mathsf{w}\colon \mathsf{A}.\ \mathsf{refl}_{(\mathsf{P}\,\mathsf{w})}\ (\mathsf{f}\,\mathsf{w}))\colon \mathsf{f} \sim \mathsf{f}$$

(ii). Let $\mathsf{f}, \mathsf{g}\colon \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\,\mathsf{w}$ with a proof $\mathsf{G}\colon \mathsf{f} \sim \mathsf{g}$. Then:

$$(\lambda \mathsf{w}\colon \mathsf{A}.\ (\mathsf{G}\,\mathsf{w})^{-1})\colon \mathsf{g} \sim \mathsf{f}$$

(iii). Let $\mathsf{f}, \mathsf{g}, \mathsf{h}\colon \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\,\mathsf{w}$ with proofs $\mathsf{G}\colon \mathsf{f} \sim \mathsf{g}$ and $\mathsf{J}\colon \mathsf{g} \sim \mathsf{h}$. Then:

$$(\lambda \mathsf{w}\colon \mathsf{A}.\ (\mathsf{G}\,\mathsf{w}) \cdot (\mathsf{J}\,\mathsf{w}))\colon \mathsf{f} \sim \mathsf{h}$$

□

Type $\mathsf{f} \sim \mathsf{g}$ is preserved under function composition as the following lemma shows.

Lemma 1.6.9. *Let $\mathsf{A}\colon \mathcal{U}_i$, $\mathsf{B}\colon \mathcal{U}_j$, and $\mathsf{C}\colon \mathcal{U}_k$ be types. Then, the following types are inhabited:*

(i) Left composition:

$$\prod_{\mathsf{f},\mathsf{g}:\mathsf{A}\to\mathsf{B}} \prod_{\mathsf{h}:\mathsf{B}\to\mathsf{C}} (\mathsf{f} \sim \mathsf{g}) \to (\mathsf{h} \circ \mathsf{f} \sim \mathsf{h} \circ \mathsf{g})$$

(ii) Right composition:

$$\prod_{\mathsf{f},\mathsf{g}:\mathsf{A}\to\mathsf{B}} \prod_{\mathsf{h}:\mathsf{C}\to\mathsf{A}} (\mathsf{f} \sim \mathsf{g}) \to (\mathsf{f} \circ \mathsf{h} \sim \mathsf{g} \circ \mathsf{h})$$

Proof. We prove each case as follows.

(i). Let $\mathsf{f}, \mathsf{g}\colon \mathsf{A} \to \mathsf{B}$ and $\mathsf{h}\colon \mathsf{B} \to \mathsf{C}$ with a proof $\mathsf{G}\colon \mathsf{f} \sim \mathsf{g}$. Then:[33]

$$(\lambda \mathsf{w}\colon \mathsf{A}.\ \mathsf{ap}_{\mathsf{h}}\ (\mathsf{G}\,\mathsf{w}))\colon \mathsf{h} \circ \mathsf{f} \sim \mathsf{h} \circ \mathsf{g}$$

[33]Here, ap is the application operator, see Lemma 1.5.62.

because by definition:

$$(\mathsf{h} \circ \mathsf{f} \sim \mathsf{h} \circ \mathsf{g}) \equiv \left(\prod_{\mathsf{w}:\mathsf{A}} (\mathsf{h} \circ \mathsf{f})\ \mathsf{w} = (\mathsf{h} \circ \mathsf{g})\ \mathsf{w}\right) \equiv \left(\prod_{\mathsf{w}:\mathsf{A}} \mathsf{h}\ (\mathsf{f}\ \mathsf{w}) = \mathsf{h}\ (\mathsf{g}\ \mathsf{w})\right)$$

(ii). Let $\mathsf{f}, \mathsf{g} : \mathsf{A} \to \mathsf{B}$ and $\mathsf{h} : \mathsf{C} \to \mathsf{A}$ with a proof $\mathsf{G} : \mathsf{f} \sim \mathsf{g}$. Then:

$$(\lambda \mathsf{w} : \mathsf{C}.\ \mathsf{G}\ (\mathsf{h}\ \mathsf{w})) : \mathsf{f} \circ \mathsf{h} \sim \mathsf{g} \circ \mathsf{h}$$

because by definition:

$$(\mathsf{f} \circ \mathsf{h} \sim \mathsf{g} \circ \mathsf{h}) \equiv \left(\prod_{\mathsf{w}:\mathsf{C}} (\mathsf{f} \circ \mathsf{h})\ \mathsf{w} = (\mathsf{g} \circ \mathsf{h})\ \mathsf{w}\right) \equiv \left(\prod_{\mathsf{w}:\mathsf{C}} \mathsf{f}\ (\mathsf{h}\ \mathsf{w}) = \mathsf{g}\ (\mathsf{h}\ \mathsf{w})\right)$$

□

We are ready to introduce the concept of equivalence. An equivalence will be a function that has a right and a left inverse.

Definition 1.6.10 (Equivalence)**.** Let $\mathsf{A} : \mathcal{U}_i$, $\mathsf{B} : \mathcal{U}_j$ be types, and $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ a function. We define type:

$$\mathsf{IsEquiv}\ \mathsf{f} :\equiv \left(\sum_{\mathsf{g}:\mathsf{B}\to\mathsf{A}} \mathsf{f} \circ \mathsf{g} \sim \mathsf{id}_{\mathsf{B}}\right) \times \left(\sum_{\mathsf{h}:\mathsf{B}\to\mathsf{A}} \mathsf{h} \circ \mathsf{f} \sim \mathsf{id}_{\mathsf{A}}\right)$$

When type $\mathsf{IsEquiv}\ \mathsf{f}$ is inhabited, we say that f is an *equivalence*. Hence, f is an equivalence if there is[34] a function g such that it is a *right inverse* for f, *and* there is a function h such that it is a *left inverse* for f.

▲

Of course, the right and left "inverses" are only up to pointwise identifiability, because the definition uses $\sim$ instead of the identity type. Nevertheless, once the function extensionality axiom is introduced, it will not matter if $\sim$ or $=$ is used in the definition of equivalence.

Convention 1.6.11. Given a term $((\mathsf{g}, \alpha), (\mathsf{h}, \beta)) : \mathsf{IsEquiv}\ \mathsf{f}$, we will write it as $(\mathsf{g}, \alpha, \mathsf{h}, \beta) : \mathsf{IsEquiv}\ \mathsf{f}$.

▲

Definition 1.6.12 (Equivalence inverse)**.** Let $\mathsf{A} : \mathcal{U}_i$, $\mathsf{B} : \mathcal{U}_j$ be types, and $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ a function. If we are given a proof $(\mathsf{g}, \alpha, \mathsf{h}, \beta) : \mathsf{IsEquiv}\ \mathsf{f}$ that f is an equivalence, then we will say that g is the *inverse* of f. The inverse of f will be denoted as f^{-1}.

▲

We have the following remark regarding the definition of inverse.

Remark 1.6.13. Notice that Definition 1.6.12 is implicitly assuming that f has a unique inverse, because it defines function g to be "the" inverse for f. We can justify that g is unique as follows.

Type $\mathsf{IsEquiv}\ \mathsf{f}$ has the property that any two terms $\mathsf{e}_1, \mathsf{e}_2 : \mathsf{IsEquiv}\ \mathsf{f}$ are identifiable, i.e. $\mathsf{e}_1 = \mathsf{e}_2$ is inhabited. This property of $\mathsf{IsEquiv}\ \mathsf{f}$ will not be proved on this report, but a proof can be found in Section 4.3 of the HoTT book [20].

This means that if we have two terms $((\mathsf{g}_1, \alpha_1), (\mathsf{h}_1, \beta_1))$, $((\mathsf{g}_2, \alpha_2), (\mathsf{h}_2, \beta_2))$ of type $\mathsf{IsEquiv}\ \mathsf{f}$, then there is a proof $\mathsf{p} : ((\mathsf{g}_1, \alpha_1), (\mathsf{h}_1, \beta_1)) = ((\mathsf{g}_2, \alpha_2), (\mathsf{h}_2, \beta_2))$, which means $\mathsf{ap}_{\mathsf{pr}_1}\ (\mathsf{ap}_{\mathsf{pr}_1}\ \mathsf{p}) : \mathsf{g}_1 = \mathsf{g}_2$.

Hence, we are justified in calling g *the* inverse, because it is unique up to identification.

Also, if we want to be strict on the wording of Definition 1.6.12, it should have been written as: "If we are given a proof $\mathsf{p} : \mathsf{IsEquiv}\ \mathsf{f}$ that f is an equivalence, then we will say that $\mathsf{pr}_1\ (\mathsf{pr}_1\ \mathsf{p})$ is the inverse of f".

However, this wording looks ugly due to the excessive use of projection functions. We are justified in writing $(\mathsf{g}, \alpha, \mathsf{h}, \beta) : \mathsf{IsEquiv}\ \mathsf{f}$ instead of $\mathsf{p} : \mathsf{IsEquiv}\ \mathsf{f}$ by Convention 1.5.27.

▲

[34] An existential in the constructive sense, see Remark 1.5.30.

It is possible to choose h as the inverse, instead of g, due to the following lemma.

Lemma 1.6.14. *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$ *be types, and* $\mathsf{f}:\mathsf{A}\to\mathsf{B}$ *a function. If we are given a proof* $(\mathsf{g},\alpha,\mathsf{h},\beta):\mathsf{IsEquiv}\,\mathsf{f}$*, then the following type is inhabited:*

$$\mathsf{g}\sim\mathsf{h}$$

Proof. By hypothesis we have $\alpha:\mathsf{f}\circ\mathsf{g}\sim\mathsf{id}_\mathsf{B}$. Then, by Lemma 1.6.9(i), there is a proof $\mathsf{p}_1:\mathsf{h}\circ(\mathsf{f}\circ\mathsf{g})\sim\mathsf{h}\circ\mathsf{id}_\mathsf{B}$,[35] which simplifies to $\mathsf{p}_1:\mathsf{h}\circ(\mathsf{f}\circ\mathsf{g})\sim\mathsf{h}$ by Lemma 1.6.4.

Similarly, we have $\beta:\mathsf{h}\circ\mathsf{f}\sim\mathsf{id}_\mathsf{A}$. Then, by Lemma 1.6.9(ii), there is a proof $\mathsf{p}_2:(\mathsf{h}\circ\mathsf{f})\circ\mathsf{g}\sim\mathsf{id}_\mathsf{A}\circ\mathsf{g}$, which simplifies to $\mathsf{p}_2:(\mathsf{h}\circ\mathsf{f})\circ\mathsf{g}\sim\mathsf{g}$ by Lemma 1.6.4. But this simplifies again to $\mathsf{p}_2:\mathsf{h}\circ(\mathsf{f}\circ\mathsf{g})\sim\mathsf{g}$ by Lemma 1.6.3.

Since we have $\mathsf{p}_2:\mathsf{h}\circ(\mathsf{f}\circ\mathsf{g})\sim\mathsf{g}$, by Lemma 1.6.8(ii) there is a proof $\mathsf{p}_3:\mathsf{g}\sim\mathsf{h}\circ(\mathsf{f}\circ\mathsf{g})$.

Finally, since we have $\mathsf{p}_3:\mathsf{g}\sim\mathsf{h}\circ(\mathsf{f}\circ\mathsf{g})$ and $\mathsf{p}_1:\mathsf{h}\circ(\mathsf{f}\circ\mathsf{g})\sim\mathsf{h}$, by Lemma 1.6.8(iii) there is a proof $\mathsf{p}_4:\mathsf{g}\sim\mathsf{h}$. □

The inverse f^{-1} behaves as expected.

Lemma 1.6.15. *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$ *be types, and* $\mathsf{f}:\mathsf{A}\to\mathsf{B}$ *a function. If* f *is an equivalence, then the following types are inhabited:*

$$\mathsf{f}\circ\mathsf{f}^{-1}\sim\mathsf{id}_\mathsf{B}$$
$$\mathsf{f}^{-1}\circ\mathsf{f}\sim\mathsf{id}_\mathsf{A}$$

Proof. Since f is an equivalence, we have $(\mathsf{f}^{-1},\alpha,\mathsf{h},\beta):\mathsf{IsEquiv}\,\mathsf{f}$.

The first type follows by hypothesis $\alpha:\mathsf{f}\circ\mathsf{f}^{-1}\sim\mathsf{id}_\mathsf{B}$.

Now, we prove the second type. By Lemma 1.6.14, there is a proof $\mathsf{p}_1:\mathsf{f}^{-1}\sim\mathsf{h}$. Then, by Lemma 1.6.9(ii), there is a proof $\mathsf{p}_2:\mathsf{f}^{-1}\circ\mathsf{f}\sim\mathsf{h}\circ\mathsf{f}$.

Since we have $\mathsf{p}_2:\mathsf{f}^{-1}\circ\mathsf{f}\sim\mathsf{h}\circ\mathsf{f}$ and $\beta:\mathsf{h}\circ\mathsf{f}\sim\mathsf{id}_\mathsf{A}$, by Lemma 1.6.8(iii) there is a proof $\mathsf{p}_3:\mathsf{f}^{-1}\circ\mathsf{f}\sim\mathsf{id}_\mathsf{A}$. □

Also, to prove that f is an equivalence, it is enough to build an inverse for it.

Lemma 1.6.16. *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$ *be types, and* $\mathsf{f}:\mathsf{A}\to\mathsf{B}$ *a function.*

If we can construct a function $\mathsf{g}:\mathsf{B}\to\mathsf{A}$ *such that the following types are inhabited:*

$$\mathsf{f}\circ\mathsf{g}\sim\mathsf{id}_\mathsf{B}$$
$$\mathsf{g}\circ\mathsf{f}\sim\mathsf{id}_\mathsf{A}$$

Then, f *is an equivalence.*

Proof. We are given $\mathsf{g}:\mathsf{B}\to\mathsf{A}$ with $\alpha:\mathsf{f}\circ\mathsf{g}\sim\mathsf{id}_\mathsf{B}$ and $\beta:\mathsf{g}\circ\mathsf{f}\sim\mathsf{id}_\mathsf{A}$ as hypotheses.

Therefore, $(\mathsf{g},\alpha,\mathsf{g},\beta):\mathsf{IsEquiv}\,\mathsf{f}$.

□

We can now define equivalence of types.

Definition 1.6.17 (Type equivalence)**.** Let $\mathsf{A}:\mathcal{U}_i$ and $\mathsf{B}:\mathcal{U}_j$ be types. We define type:

$$\mathsf{A}\simeq\mathsf{B}:\equiv\sum_{\mathsf{f}:\mathsf{A}\to\mathsf{B}}\mathsf{IsEquiv}\,\mathsf{f}$$

When type $\mathsf{A}\simeq\mathsf{B}$ is inhabited, we say that A and B are *equivalent types*. Hence, two types are equivalent when there is an equivalence between them.

▲

[35] In more detail, by Lemma 1.6.9(i), there is a term:

$$\mathsf{G}:\prod_{\mathsf{j},\mathsf{k}:\mathsf{B}\to\mathsf{B}}\ \prod_{\mathsf{m}:\mathsf{B}\to\mathsf{A}}(\mathsf{j}\sim\mathsf{k})\to(\mathsf{m}\circ\mathsf{j}\sim\mathsf{m}\circ\mathsf{k})$$

Therefore, $\mathsf{G}\ (\mathsf{f}\circ\mathsf{g})\ \mathsf{id}_\mathsf{B}\ \mathsf{h}\ \alpha:\mathsf{h}\circ(\mathsf{f}\circ\mathsf{g})\sim\mathsf{h}\circ\mathsf{id}_\mathsf{B}$.

To prove a type equivalence, it suffices to construct two functions between the types, such that these functions are inverses of each other, as the following lemma shows.

Lemma 1.6.18. *Let* $\mathsf{A}:\mathcal{U}_i$ *and* $\mathsf{B}:\mathcal{U}_j$ *be types.*
If we have two functions $\mathsf{f}:\mathsf{A}\to\mathsf{B}$ *and* $\mathsf{g}:\mathsf{B}\to\mathsf{A}$ *such that the following types are inhabited:*

$$\mathsf{f}\circ\mathsf{g}\sim\mathsf{id}_{\mathsf{B}}$$
$$\mathsf{g}\circ\mathsf{f}\sim\mathsf{id}_{\mathsf{A}}$$

Then, types A *and* B *are equivalent.*

Proof. We are given $\mathsf{f}:\mathsf{A}\to\mathsf{B}$ and $\mathsf{g}:\mathsf{B}\to\mathsf{A}$ with $\alpha:\mathsf{f}\circ\mathsf{g}\sim\mathsf{id}_{\mathsf{B}}$ and $\beta:\mathsf{g}\circ\mathsf{f}\sim\mathsf{id}_{\mathsf{A}}$ as hypotheses.
Therefore, $(\mathsf{g},\alpha,\mathsf{g},\beta):\mathsf{IsEquiv}\,\mathsf{f}$, which means $(\mathsf{f},(\mathsf{g},\alpha,\mathsf{g},\beta)):\mathsf{A}\simeq\mathsf{B}$. □

Now, we present a series of results involving equivalences.

Lemma 1.6.19 (Reflexivity of equivalences)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *be a type. Then* $\mathsf{id}_{\mathsf{A}}:\mathsf{A}\to\mathsf{A}$ *is an equivalence. Hence the following type is inhabited:*

$$\mathsf{A}\simeq\mathsf{A}$$

Proof. We use Lemma 1.6.16. We need to define a function acting as inverse for id_{A}. Just take id_{A} itself. Now, we have to prove:

$$\mathsf{id}_{\mathsf{A}}\circ\mathsf{id}_{\mathsf{A}}\sim\mathsf{id}_{\mathsf{A}}$$
$$\mathsf{id}_{\mathsf{A}}\circ\mathsf{id}_{\mathsf{A}}\sim\mathsf{id}_{\mathsf{A}}$$

But, by Lemma 1.6.4, type $\mathsf{id}_{\mathsf{A}}\circ\mathsf{id}_{\mathsf{A}}\sim\mathsf{id}_{\mathsf{A}}$ simplifies to $\mathsf{id}_{\mathsf{A}}\sim\mathsf{id}_{\mathsf{A}}$, which is always inhabited (Lemma 1.6.8(i)). □

Lemma 1.6.20 (Symmetry of equivalences)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *and* $\mathsf{B}:\mathcal{U}_j$ *be types. If* $\mathsf{e}:\mathsf{A}\to\mathsf{B}$ *is an equivalence, then* $\mathsf{e}^{-1}:\mathsf{B}\to\mathsf{A}$ *is an equivalence.*
Hence, the following type is inhabited:

$$(\mathsf{A}\simeq\mathsf{B})\to(\mathsf{B}\simeq\mathsf{A})$$

Proof. Since e is an equivalence, by Lemma 1.6.15, the following types are inhabited:

$$\mathsf{e}\circ\mathsf{e}^{-1}\sim\mathsf{id}_{\mathsf{B}}$$
$$\mathsf{e}^{-1}\circ\mathsf{e}\sim\mathsf{id}_{\mathsf{A}}$$

But, by Lemma 1.6.16, this means that e^{-1} is also an equivalence. □

Lemma 1.6.21 (Transitivity of equivalences)**.** *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$, *and* $\mathsf{C}:\mathcal{U}_k$ *be types. If* $\mathsf{e}_1:\mathsf{A}\to\mathsf{B}$ *and* $\mathsf{e}_2:\mathsf{B}\to\mathsf{C}$ *are equivalences, then* $\mathsf{e}_2\circ\mathsf{e}_1:\mathsf{A}\to\mathsf{C}$ *is an equivalence.*
Hence, the following type is inhabited:

$$(\mathsf{A}\simeq\mathsf{B})\to(\mathsf{B}\simeq\mathsf{C})\to(\mathsf{A}\simeq\mathsf{C})$$

Proof. Since e_1 and e_2 are equivalences, by Lemma 1.6.15, the following types are inhabited:

$$\mathsf{e}_1\circ\mathsf{e}_1^{-1}\sim\mathsf{id}_{\mathsf{B}}$$
$$\mathsf{e}_1^{-1}\circ\mathsf{e}_1\sim\mathsf{id}_{\mathsf{A}}$$
$$\mathsf{e}_2\circ\mathsf{e}_2^{-1}\sim\mathsf{id}_{\mathsf{C}}$$
$$\mathsf{e}_2^{-1}\circ\mathsf{e}_2\sim\mathsf{id}_{\mathsf{B}}$$

We will use Lemma 1.6.16 to conclude that $\mathsf{e}_2\circ\mathsf{e}_1$ is an equivalence. We propose function $\mathsf{e}_1^{-1}\circ\mathsf{e}_2^{-1}$ as the inverse for $\mathsf{e}_2\circ\mathsf{e}_1$.

By Lemma 1.6.9(i), type $\mathsf{e}_2 \circ (\mathsf{e}_1 \circ \mathsf{e}_1^{-1}) \sim \mathsf{e}_2 \circ \mathsf{id}_B$ is inhabited, which simplifies to $\mathsf{e}_2 \circ (\mathsf{e}_1 \circ \mathsf{e}_1^{-1}) \sim \mathsf{e}_2$ by Lemma 1.6.4.

Now, by Lemma 1.6.9(ii), type $(\mathsf{e}_2 \circ (\mathsf{e}_1 \circ \mathsf{e}_1^{-1})) \circ \mathsf{e}_2^{-1} \sim \mathsf{e}_2 \circ \mathsf{e}_2^{-1}$ is inhabited, which simplifies to $(\mathsf{e}_2 \circ \mathsf{e}_1) \circ (\mathsf{e}_1^{-1} \circ \mathsf{e}_2^{-1}) \sim \mathsf{e}_2 \circ \mathsf{e}_2^{-1}$ by Lemma 1.6.3.

But $\mathsf{e}_2 \circ \mathsf{e}_2^{-1} \sim \mathsf{id}_C$, hence, by Lemma 1.6.8(iii), the first equation follows:

$$(\mathsf{e}_2 \circ \mathsf{e}_1) \circ (\mathsf{e}_1^{-1} \circ \mathsf{e}_2^{-1}) \sim \mathsf{id}_C$$

The second equation:

$$(\mathsf{e}_1^{-1} \circ \mathsf{e}_2^{-1}) \circ (\mathsf{e}_2 \circ \mathsf{e}_1) \sim \mathsf{id}_A$$

will hold by an identical strategy.

□

The next property expresses that identifiable elements produce equivalent properties.

Lemma 1.6.22 (Transport is an equivalence)**.** *Let* $\mathsf{A} : \mathcal{U}_i$ *be a type and* $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$ *a type family. Then the following type is inhabited:*

$$\prod_{w,y:A} \prod_{q:w=y} \mathsf{IsEquiv}\ (\mathsf{transport}^{\mathsf{P}}\ q)$$

Hence, the following type is also inhabited:

$$\prod_{w,y:A} (w = y) \to ((\mathsf{P}\ w) \simeq (\mathsf{P}\ y)) \tag{1.15}$$

Proof. By the induction principle for identity types, it is enough to prove that the following type is inhabited:

$$\prod_{a:A} \mathsf{IsEquiv}\ (\mathsf{transport}^{\mathsf{P}}\ (\mathsf{refl}_{\mathsf{A}}\ a))$$

But $\mathsf{transport}^{\mathsf{P}}\ (\mathsf{refl}_{\mathsf{A}}\ a) \equiv \mathsf{id}_{(\mathsf{P}\ a)}$ by Lemma 1.5.61. Hence, it is enough to prove:

$$\prod_{a:A} \mathsf{IsEquiv}\ \mathsf{id}_{(\mathsf{P}\ a)}$$

which follows by Lemma 1.6.19.

Now, type (1.15) is inhabited, because given $w, y : \mathsf{A}$ and $q : w = y$, we proved above that:

$$\mathsf{transport}^{\mathsf{P}}\ q : (\mathsf{P}\ w) \to (\mathsf{P}\ y)$$

is an equivalence. Therefore, $(\mathsf{P}\ w) \simeq (\mathsf{P}\ y)$ is inhabited by definition of type equivalence.

□

The following property expresses that parentheses in nested pairs can be rearranged.

Lemma 1.6.23 (Associativity of Σ-types)**.** *Let* $\mathsf{A} : \mathcal{U}_i$ *be a type, and* $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$, $\mathsf{Q} : (\sum_{w:A} \mathsf{P}\ w) \to \mathcal{U}_k$ *type families.*
If we define the family:

$$\mathsf{T} :\equiv \left(\lambda w : \mathsf{A}.\ \sum_{y:\mathsf{P}\ w} \mathsf{Q}\ (w, y)\right) : \mathsf{A} \to \mathcal{U}_{\max\{j,k\}}$$

then there is a function:

$$\mathsf{m} : \prod_{y:\sum_{w:A} \mathsf{P}\ w} (\mathsf{Q}\ y) \to (\mathsf{T}\ (\mathsf{pr}_1\ y))$$

such that the following function is an equivalence:[36]

$$\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m} : \left(\sum_{q:\sum_{w:A} \mathsf{P}\ w} \mathsf{Q}\ q\right) \to \left(\sum_{w:A} \mathsf{T}\ w\right)$$

[36] Function Σ_{map} was defined in Lemma 1.5.28.

or equivalently (by the definition of T*):*

$$\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\colon \left(\sum_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{Q}\ \mathsf{q}\right) \to \left(\sum_{\mathsf{w}:\mathsf{A}}\ \sum_{\mathsf{y}:\mathsf{P}\ \mathsf{w}} \mathsf{Q}\ (\mathsf{w},\mathsf{y})\right)$$

Hence, the following type is inhabited:

$$\left(\sum_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{Q}\ \mathsf{q}\right) \simeq \left(\sum_{\mathsf{w}:\mathsf{A}}\ \sum_{\mathsf{y}:\mathsf{P}\ \mathsf{w}} \mathsf{Q}\ (\mathsf{w},\mathsf{y})\right)$$

Proof. We will construct function m using the induction principle for Σ-types. So, it is enough to define a function:

$$\mathsf{g}\colon \prod_{\mathsf{w}:\mathsf{A}}\ \prod_{\mathsf{y}:\mathsf{P}\ \mathsf{w}} (\mathsf{Q}\ (\mathsf{w},\mathsf{y})) \to (\mathsf{T}\ (\mathsf{pr}_1\ (\mathsf{w},\mathsf{y})))$$

or equivalently (by definition of T and simplification):

$$\mathsf{g}\colon \prod_{\mathsf{w}:\mathsf{A}}\ \prod_{\mathsf{y}:\mathsf{P}\ \mathsf{w}} (\mathsf{Q}\ (\mathsf{w},\mathsf{y})) \to \left(\sum_{\mathsf{t}:\mathsf{P}\ \mathsf{w}} \mathsf{Q}\ (\mathsf{w},\mathsf{t})\right)$$

Define:

$$\mathsf{g} :\equiv \lambda(\mathsf{w}:\mathsf{A})(\mathsf{y}:\mathsf{P}\ \mathsf{w})(\mathsf{q}:\mathsf{Q}\ (\mathsf{w},\mathsf{y}).\ (\mathsf{y},\mathsf{q})$$

Hence, by the induction principle, we have (for arbitrary $\mathsf{a}:\mathsf{A}$ and $\mathsf{b}:\mathsf{P}\ \mathsf{a}$):

$$\mathsf{m}\ (\mathsf{a},\mathsf{b}) \equiv \mathsf{g}\ \mathsf{a}\ \mathsf{b}$$

To prove that $\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}$ is an equivalence, we use Lemma 1.6.16. To define its inverse function, we will use the recursion principle for Σ-types. So, to construct:

$$\mathsf{inv}\colon \left(\sum_{\mathsf{w}:\mathsf{A}}\ \sum_{\mathsf{y}:\mathsf{P}\ \mathsf{w}} \mathsf{Q}\ (\mathsf{w},\mathsf{y})\right) \to \left(\sum_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{Q}\ \mathsf{q}\right)$$

just define it on pairs:

$$\mathsf{inv}\ (\mathsf{a},\mathsf{b}) :\equiv ((\mathsf{a},\mathsf{pr}_1\ \mathsf{b}),\mathsf{pr}_2\ \mathsf{b})$$

where $\mathsf{a}:\mathsf{A}$ and $\mathsf{b}:\sum_{\mathsf{y}:\mathsf{P}\ \mathsf{a}} \mathsf{Q}\ (\mathsf{a},\mathsf{y})$. Notice this is well-typed, since $\mathsf{pr}_1\ \mathsf{b}:\mathsf{P}\ \mathsf{a}$ and $\mathsf{pr}_2\ \mathsf{b}:\mathsf{Q}\ (\mathsf{a},\mathsf{pr}_1\ \mathsf{b})$.

Now, we need to check that $\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}$ and inv are inverses of each other.

- $\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\ (\mathsf{inv}\ \mathsf{p}) = \mathsf{p}$ for arbitrary $\mathsf{p}:\sum_{\mathsf{w}:\mathsf{A}} \sum_{\mathsf{y}:\mathsf{P}\ \mathsf{w}} \mathsf{Q}\ (\mathsf{w},\mathsf{y})$.

  By the induction principle for Σ-types, it is enough to prove that the following type is inhabited:

$$\prod_{\mathsf{a}:\mathsf{A}}\ \prod_{\mathsf{b}:\sum_{\mathsf{y}:\mathsf{P}\ \mathsf{a}} \mathsf{Q}\ (\mathsf{a},\mathsf{y})} \Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\ (\mathsf{inv}\ (\mathsf{a},\mathsf{b})) = (\mathsf{a},\mathsf{b})$$

  So, suppose $\mathsf{a}:\mathsf{A}$. But:

$$\prod_{\mathsf{b}:\sum_{\mathsf{y}:\mathsf{P}\ \mathsf{a}} \mathsf{Q}\ (\mathsf{a},\mathsf{y})} \Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\ (\mathsf{inv}\ (\mathsf{a},\mathsf{b})) = (\mathsf{a},\mathsf{b})$$

  can be proved by the induction principle again. Therefore, it is enough to prove:

$$\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\ (\mathsf{inv}\ (\mathsf{a},(\mathsf{c},\mathsf{d}))) = (\mathsf{a},(\mathsf{c},\mathsf{d}))$$

for arbitrary $\mathsf{c} : \mathsf{P}\ \mathsf{a}$ and $\mathsf{d} : \mathsf{Q}\ (\mathsf{a}, \mathsf{c})$. Hence:

$$\begin{aligned}\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\ (\mathsf{inv}\ (\mathsf{a}, (\mathsf{c}, \mathsf{d}))) &\equiv \Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\ ((\mathsf{a}, \mathsf{pr}_1\ (\mathsf{c}, \mathsf{d})), \mathsf{pr}_2\ (\mathsf{c}, \mathsf{d})) \\ &\equiv \Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\ ((\mathsf{a}, \mathsf{c}), \mathsf{d}) \\ &\equiv (\mathsf{pr}_1\ (\mathsf{a}, \mathsf{c}), \mathsf{m}\ (\mathsf{a}, \mathsf{c})\ \mathsf{d}) \\ &\equiv (\mathsf{a}, \mathsf{g}\ \mathsf{a}\ \mathsf{c}\ \mathsf{d}) \\ &\equiv (\mathsf{a}, (\mathsf{c}, \mathsf{d}))\end{aligned}$$

- $\mathsf{inv}\ (\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\ \mathsf{p}) = \mathsf{p}$ for arbitrary $\mathsf{p} : \sum_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{Q}\ \mathsf{q}$.

  By the induction principle for Σ-types applied *twice*, it is enough to prove that the following type is inhabited:

$$\prod_{\mathsf{a}:\mathsf{A}} \prod_{\mathsf{c}:\mathsf{P}\ \mathsf{a}} \prod_{\mathsf{d}:\mathsf{Q}\ (\mathsf{a},\mathsf{c})} \mathsf{inv}\ (\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\ ((\mathsf{a}, \mathsf{c}), \mathsf{d})) = ((\mathsf{a}, \mathsf{c}), \mathsf{d})$$

  Hence:

$$\begin{aligned}\mathsf{inv}\ (\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{m}\ ((\mathsf{a}, \mathsf{c}), \mathsf{d})) &\equiv \mathsf{inv}\ (\mathsf{pr}_1\ (\mathsf{a}, \mathsf{c}), \mathsf{m}\ (\mathsf{a}, \mathsf{c})\ \mathsf{d}) \\ &\equiv \mathsf{inv}\ (\mathsf{a}, \mathsf{g}\ \mathsf{a}\ \mathsf{c}\ \mathsf{d}) \\ &\equiv \mathsf{inv}\ (\mathsf{a}, (\mathsf{c}, \mathsf{d})) \\ &\equiv ((\mathsf{a}, \mathsf{pr}_1\ (\mathsf{c}, \mathsf{d})), \mathsf{pr}_2\ (\mathsf{c}, \mathsf{d})) \\ &\equiv ((\mathsf{a}, \mathsf{c}), \mathsf{d})\end{aligned}$$

□

Corollary 1.6.24 (Associativity of $\times$)**.** *Let $\mathsf{A} : \mathcal{U}_i$, $\mathsf{B} : \mathcal{U}_j$, and $\mathsf{C} : \mathcal{U}_k$ be types. Then, the following type is inhabited:*

$$(\mathsf{A} \times \mathsf{B}) \times \mathsf{C} \simeq \mathsf{A} \times (\mathsf{B} \times \mathsf{C})$$

Proof. Apply Lemma 1.6.23 with the constant families:

$$\mathsf{P} :\equiv (\lambda \mathsf{w}.\ \mathsf{B}) : \mathsf{A} \to \mathcal{U}_j$$
$$\mathsf{Q} :\equiv (\lambda \mathsf{q}.\ \mathsf{C}) : \left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right) \to \mathcal{U}_k$$

and simplify applications accordingly.

□

The next property states that any identity proof on a pair can be split into identity proofs on its coordinates.

Lemma 1.6.25. *Let $\mathsf{A} : \mathcal{U}_i$ be a type, and $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$ a type family. Then, the following type is inhabited:*[37]

$$\prod_{\mathsf{y},\mathsf{z}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \left((\mathsf{y} = \mathsf{z}) \simeq \sum_{\mathsf{q}:\ \mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}} \mathsf{q}_*\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}\right) \tag{1.16}$$

Proof. To prove that (1.16) is inhabited, we will have to define some auxiliary functions.

First, we define a function by induction on identity types:

$$\mathsf{f}_1 : \prod_{\mathsf{y},\mathsf{z}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \left((\mathsf{y} = \mathsf{z}) \to \left(\sum_{\mathsf{q}:\ \mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}} \mathsf{q}_*\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}\right)\right)$$

[37] Here, q_* is the transport operator, see Lemma 1.5.61. Also, notice this is well-typed, since $\mathsf{q}_* : \mathsf{P}\ (\mathsf{pr}_1\ \mathsf{y}) \to \mathsf{P}\ (\mathsf{pr}_1\ \mathsf{z})$, and $\mathsf{pr}_2\ \mathsf{y} : \mathsf{P}\ (\mathsf{pr}_1\ \mathsf{y})$, and $\mathsf{pr}_2\ \mathsf{z} : \mathsf{P}\ (\mathsf{pr}_1\ \mathsf{z})$.

by stating how it acts on reflexivity proofs for arbitrary $\mathsf{a} : \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$:

$$\mathsf{f}_1\ \mathsf{a}\ \mathsf{a}\ \left(\mathsf{refl}_{\left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right)}\ \mathsf{a}\right) :\equiv (\ \mathsf{refl}_{\mathsf{A}}\ (\mathsf{pr}_1\ \mathsf{a}),\ \mathsf{refl}_{(\mathsf{P}\ (\mathsf{pr}_1\ \mathsf{a}))}\ (\mathsf{pr}_2\ \mathsf{a})\ )$$

Notice this equation is well-typed, since:

$$\mathsf{refl}_{\mathsf{A}}\ (\mathsf{pr}_1\ \mathsf{a}) : \mathsf{pr}_1\ \mathsf{a} = \mathsf{pr}_1\ \mathsf{a}$$

and:

$$\mathsf{refl}_{(\mathsf{P}\ (\mathsf{pr}_1\ \mathsf{a}))}\ (\mathsf{pr}_2\ \mathsf{a}) : \mathsf{pr}_2\ \mathsf{a} = \mathsf{pr}_2\ \mathsf{a}$$

or equivalently:

$$\mathsf{refl}_{(\mathsf{P}\ (\mathsf{pr}_1\ \mathsf{a}))}\ (\mathsf{pr}_2\ \mathsf{a}) : (\mathsf{refl}_{\mathsf{A}}\ (\mathsf{pr}_1\ \mathsf{a}))_*\ \mathsf{pr}_2\ \mathsf{a} = \mathsf{pr}_2\ \mathsf{a}$$

because $(\mathsf{refl}_{\mathsf{A}}\ (\mathsf{pr}_1\ \mathsf{a}))_* \equiv \mathsf{id}_{(\mathsf{P}\ (\mathsf{pr}_1\ \mathsf{a}))}$ by definition of the transport operator in Lemma 1.5.61.

In other words, both sides of the equation will have type:

$$\sum_{\mathsf{q}:\ \mathsf{pr}_1\ \mathsf{a}=\mathsf{pr}_1\ \mathsf{a}} \mathsf{q}_*\ (\mathsf{pr}_2\ \mathsf{a}) = \mathsf{pr}_2\ \mathsf{a}$$

Now, we define another function by induction on identity types:

$$\mathsf{g}_1 : \prod_{\mathsf{a}:\mathsf{A}}\ \prod_{\mathsf{y},\mathsf{z}:\mathsf{P}\ \mathsf{a}} (\mathsf{y} = \mathsf{z}) \to ((\mathsf{a}, \mathsf{y}) = (\mathsf{a}, \mathsf{z}))$$

by stating how it acts on reflexivity proofs:[38]

$$\mathsf{g}_1\ \mathsf{a}\ \mathsf{b}\ \mathsf{b}\ (\mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ \mathsf{b}) :\equiv \mathsf{refl}_{\left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right)}\ (\mathsf{a}, \mathsf{b})$$

for arbitrary $\mathsf{a} : \mathsf{A}$ and $\mathsf{b} : \mathsf{P}\ \mathsf{a}$.

We define yet another function by induction on identity types:

$$\mathsf{g}_2 : \prod_{\mathsf{y}_1,\mathsf{z}_1:\mathsf{A}}\ \prod_{\mathsf{q}:\mathsf{y}_1=\mathsf{z}_1}\ \prod_{\mathsf{y}_2:\mathsf{P}\ \mathsf{y}_1}\ \prod_{\mathsf{z}_2:\mathsf{P}\ \mathsf{z}_1} (\mathsf{q}_*\ \mathsf{y}_2 = \mathsf{z}_2) \to ((\mathsf{y}_1, \mathsf{y}_2) = (\mathsf{z}_1, \mathsf{z}_2))$$

as:

$$\mathsf{g}_2\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) :\equiv \mathsf{g}_1\ \mathsf{a}$$

for arbitrary $\mathsf{a} : \mathsf{A}$.

Notice this equation is well-typed, since $\mathsf{g}_2\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})$ has type:

$$\prod_{\mathsf{y}_2:\mathsf{P}\ \mathsf{a}}\ \prod_{\mathsf{z}_2:\mathsf{P}\ \mathsf{a}} ((\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})_*\ \mathsf{y}_2 = \mathsf{z}_2) \to ((\mathsf{a}, \mathsf{y}_2) = (\mathsf{a}, \mathsf{z}_2))$$

which simplifies to:

$$\prod_{\mathsf{y}_2,\mathsf{z}_2:\mathsf{P}\ \mathsf{a}} (\mathsf{y}_2 = \mathsf{z}_2) \to ((\mathsf{a}, \mathsf{y}_2) = (\mathsf{a}, \mathsf{z}_2))$$

and this is precisely the type of $\mathsf{g}_1\ \mathsf{a}$.

Finally, we define a function:

$$\mathsf{g}_3 : \prod_{\mathsf{y},\mathsf{z}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \left(\left(\sum_{\mathsf{q}:\ \mathsf{pr}_1\ \mathsf{y}=\mathsf{pr}_1\ \mathsf{z}} \mathsf{q}_*\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}\right) \to (\mathsf{y} = \mathsf{z})\right)$$

by induction on Σ-types applied three times:

$$\mathsf{g}_3\ (\mathsf{a}_1, \mathsf{b}_1)\ (\mathsf{a}_2, \mathsf{b}_2)\ (\mathsf{q}_1, \mathsf{q}_2) :\equiv \mathsf{g}_2\ \mathsf{a}_1\ \mathsf{a}_2\ \mathsf{q}_1\ \mathsf{b}_1\ \mathsf{b}_2\ \mathsf{q}_2$$

[38] Notice we are using implicit lambda notation on parameter a, i.e. we introduce variable $\mathsf{a} : \mathsf{A}$ and *then* we apply the induction principle.

where $a_1 : A$, $b_1 : P\ a_1$, $a_2 : A$, $b_2 : P\ a_2$, $q_1 : pr_1\ (a_1, b_1) = pr_1\ (a_2, b_2)$, or equivalently, $q_1 : a_1 = a_2$, and $q_2 : q_{1*}\ (pr_2\ (a_1, b_1)) = pr_2\ (a_2, b_2)$, or equivalently, $q_2 : q_{1*}\ b_1 = b_2$.

Now, we prove the following claims.

Claim 1: The following type is inhabited:

$$\prod_{y,z:\sum_{w:A} P\ w}\ \prod_{q:y=z} g_3\ y\ z\ (f_1\ y\ z\ q) = q$$

By the induction principle for identity types, it is enough to prove:

$$\prod_{a:\sum_{w:A} P\ w} g_3\ a\ a\ (f_1\ a\ a\ (\mathsf{refl}_{\left(\sum_{w:A} P\ w\right)}\ a)) = \mathsf{refl}_{\left(\sum_{w:A} P\ w\right)}\ a$$

But then, by the induction principle for Σ-types, it is enough to prove, for arbitrary $c : A$ and $d : P\ c$, the following:

$$g_3\ (c, d)\ (c, d)\ (f_1\ (c, d)\ (c, d)\ (\mathsf{refl}_{\left(\sum_{w:A} P\ w\right)}\ (c, d))) = \mathsf{refl}_{\left(\sum_{w:A} P\ w\right)}\ (c, d)$$

Hence:

$$\begin{aligned}
&g_3\ (c, d)\ (c, d)\ (f_1\ (c, d)\ (c, d)\ (\mathsf{refl}_{\left(\sum_{w:A} P\ w\right)}\ (c, d))) \\
\equiv\ &g_3\ (c, d)\ (c, d)\ (\ \mathsf{refl}_A\ (pr_1\ (c, d)),\ \mathsf{refl}_{(P\ (pr_1\ (c,d)))}\ (pr_2\ (c, d))\ ) \\
\equiv\ &g_3\ (c, d)\ (c, d)\ (\mathsf{refl}_A\ c,\ \mathsf{refl}_{(P\ c)}\ d) \\
\equiv\ &g_2\ c\ c\ (\mathsf{refl}_A\ c)\ d\ d\ (\mathsf{refl}_{(P\ c)}\ d) \\
\equiv\ &g_1\ c\ d\ d\ (\mathsf{refl}_{(P\ c)}\ d) \\
\equiv\ &\mathsf{refl}_{\left(\sum_{w:A} P\ w\right)}\ (c, d)
\end{aligned}$$

This proves the claim.

Claim 2: The following type is inhabited:

$$\prod_{y,z:\sum_{w:A} P\ w}\ \prod_{t:\sum_{q:\,pr_1\ y=pr_1\ z}\ \prod_{q_*\ (pr_2\ y)=pr_2\ z}} f_1\ y\ z\ (g_3\ y\ z\ t) = t$$

By applying three times the induction principle for Σ-types (and simplifying), it is enough to prove:

$$\prod_{a_1:A}\ \prod_{b_1:P\ a_1}\ \prod_{a_2:A}\ \prod_{b_2:P\ a_2}\ \prod_{q_1:a_1=a_2}\ \prod_{q_2:q_{1*}\ b_1=b_2} f_1\ (a_1, b_1)\ (a_2, b_2)\ (g_3\ (a_1, b_1)\ (a_2, b_2)\ (q_1, q_2)) = (q_1, q_2) \tag{1.17}$$

But it is enough to prove:[39]

$$\prod_{a_1,a_2:A}\ \prod_{q_1:a_1=a_2}\ \prod_{b_1:P\ a_1}\ \prod_{b_2:P\ a_2}\ \prod_{q_2:q_{1*}\ b_1=b_2} f_1\ (a_1, b_1)\ (a_2, b_2)\ (g_3\ (a_1, b_1)\ (a_2, b_2)\ (q_1, q_2)) = (q_1, q_2) \tag{1.18}$$

Therefore, by the induction principle for identities types, it is enough to prove (after simplification):

$$\prod_{a:A}\ \prod_{b_1:P\ a}\ \prod_{b_2:P\ a}\ \prod_{q_2:b_1=b_2} f_1\ (a, b_1)\ (a, b_2)\ (g_3\ (a, b_1)\ (a, b_2)\ (\mathsf{refl}_A\ a, q_2)) = (\mathsf{refl}_A\ a, q_2)$$

[39] If (1.18) has a proof G, then:

$$\lambda a_1.\ \lambda b_1.\ \lambda a_2.\ \lambda b_2.\ \lambda q_1.\ \lambda q_2.\ G\ a_1\ a_2\ q_1\ b_1\ b_2\ q_2$$

will be a proof for (1.17).

So, suppose $\mathsf{a} : \mathsf{A}$. But then, by the induction principle for identity types, it is enough to prove:

$$\prod_{\mathsf{b}:\mathsf{P}\ \mathsf{a}} \mathsf{f}_1\ (\mathsf{a},\mathsf{b})\ (\mathsf{a},\mathsf{b})\ (\mathsf{g}_3\ (\mathsf{a},\mathsf{b})\ (\mathsf{a},\mathsf{b})\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}, \mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ \mathsf{b})) = (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}, \mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ \mathsf{b})$$

Hence:

$$\begin{aligned}
&\mathsf{f}_1\ (\mathsf{a},\mathsf{b})\ (\mathsf{a},\mathsf{b})\ (\mathsf{g}_3\ (\mathsf{a},\mathsf{b})\ (\mathsf{a},\mathsf{b})\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}, \mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ \mathsf{b})) \\
&\equiv \mathsf{f}_1\ (\mathsf{a},\mathsf{b})\ (\mathsf{a},\mathsf{b})\ (\mathsf{g}_2\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\ \mathsf{b}\ \mathsf{b}\ (\mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ \mathsf{b})) \\
&\equiv \mathsf{f}_1\ (\mathsf{a},\mathsf{b})\ (\mathsf{a},\mathsf{b})\ (\mathsf{g}_1\ \mathsf{a}\ \mathsf{b}\ \mathsf{b}\ (\mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ \mathsf{b})) \\
&\equiv \mathsf{f}_1\ (\mathsf{a},\mathsf{b})\ (\mathsf{a},\mathsf{b})\ (\mathsf{refl}_{\left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right)}\ (\mathsf{a},\mathsf{b})) \\
&\equiv (\ \mathsf{refl}_{\mathsf{A}}\ (\mathsf{pr}_1\ (\mathsf{a},\mathsf{b})),\ \mathsf{refl}_{(\mathsf{P}\ (\mathsf{pr}_1\ (\mathsf{a},\mathsf{b})))}\ (\mathsf{pr}_2\ (\mathsf{a},\mathsf{b}))\ ) \\
&\equiv (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}, \mathsf{refl}_{(\mathsf{P}\ \mathsf{a})}\ \mathsf{b})
\end{aligned}$$

This proves the claim.

Now, to prove (1.16), we will use Lemma 1.6.18. Let $\mathsf{y},\mathsf{z} : \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$. Define:

$$\mathsf{h}_1 :\equiv (\lambda \mathsf{t}.\ \mathsf{f}_1\ \mathsf{y}\ \mathsf{z}\ \mathsf{t}) : (\mathsf{y} = \mathsf{z}) \to \left(\sum_{\mathsf{q}:\ \mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}} \mathsf{q}_*\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}\right)$$

$$\mathsf{h}_2 :\equiv (\lambda \mathsf{t}.\ \mathsf{g}_3\ \mathsf{y}\ \mathsf{z}\ \mathsf{t}) : \left(\sum_{\mathsf{q}:\ \mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}} \mathsf{q}_*\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}\right) \to (\mathsf{y} = \mathsf{z})$$

The equation $\mathsf{h}_2\ (\mathsf{h}_1\ \mathsf{t}) = \mathsf{t}$ follows by Claim 1, while equation $\mathsf{h}_1\ (\mathsf{h}_2\ \mathsf{t}) = \mathsf{t}$ follows by Claim 2.

□

We have the following corollary.

Corollary 1.6.26. *Let $\mathsf{A} : \mathcal{U}_i$ and $\mathsf{B} : \mathcal{U}_j$ be types. The following type is inhabited:*

$$\prod_{\mathsf{y},\mathsf{z}:\ \mathsf{A}\times\mathsf{B}} (\mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}) \to (\mathsf{pr}_2\ \mathsf{y} = \mathsf{pr}_2\ \mathsf{z}) \to (\mathsf{y} = \mathsf{z})$$

Proof. Let $\mathsf{y},\mathsf{z} : \mathsf{A} \times \mathsf{B}$ with:

$$\mathsf{h}_1 : \mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}$$
$$\mathsf{h}_2 : \mathsf{pr}_2\ \mathsf{y} = \mathsf{pr}_2\ \mathsf{z}$$

By applying Lemma 1.6.25 with the constant family $\mathsf{P} :\equiv (\lambda \mathsf{r} : \mathsf{A}.\ \mathsf{B}) : \mathsf{A} \to \mathcal{U}_j$, it is enough to prove:

$$\sum_{\mathsf{q}:\ \mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}} \mathsf{transport}^{(\lambda \mathsf{r}.\ \mathsf{B})}\ \mathsf{q}\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}$$

We already have $\mathsf{h}_1 : \mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}$ by hypothesis. Therefore, it remains to show that the following type is inhabited:

$$\mathsf{transport}^{(\lambda \mathsf{r}.\ \mathsf{B})}\ \mathsf{h}_1\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}$$

By Lemma 1.6.6, there is a term:

$$\mathsf{p}_1 : \mathsf{transport}^{(\lambda \mathsf{r}.\ \mathsf{B})}\ \mathsf{h}_1\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{y}$$

Therefore, we have:

$$\mathsf{p}_1 \centerdot \mathsf{h}_2 : \mathsf{transport}^{(\lambda \mathsf{r}.\ \mathsf{B})}\ \mathsf{h}_1\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}$$

□

1.6.1 The function extensionality axiom

Given two functions $\mathsf{f},\mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}$, the following lemma shows that they are pointwise identifiable whenever they are identifiable.

Lemma 1.6.27. *Let $\mathsf{A}:\mathcal{U}_i$ be a type and $\mathsf{P}:\mathsf{A}\to\mathcal{U}_j$ a type family.*
Then, there is a function:

$$\mathsf{happly}:\prod_{\mathsf{f},\mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}}(\mathsf{f}=\mathsf{g})\to(\mathsf{f}\sim\mathsf{g})$$

defined by induction on identity types as:

$$\mathsf{happly}\;\mathsf{k}\;\mathsf{k}\;\left(\mathsf{refl}_{\left(\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}\right)}\;\mathsf{k}\right):\equiv\lambda\mathsf{w}:\mathsf{A}.\;\mathsf{refl}_{(\mathsf{P}\,\mathsf{w})}\;(\mathsf{k}\,\mathsf{w})$$

for arbitrary $\mathsf{k}:\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}$.

Proof. Directly by the induction principle for identity types. Just notice:

$$(\lambda\mathsf{w}:\mathsf{A}.\;\mathsf{refl}_{(\mathsf{P}\,\mathsf{w})}\;(\mathsf{k}\,\mathsf{w})):\prod_{\mathsf{w}:\mathsf{A}}\mathsf{k}\,\mathsf{w}=\mathsf{k}\,\mathsf{w}$$

□

However, as was discussed after Definition 1.6.7, in our type theory so far it is not possible to prove the converse of Lemma 1.6.27, i,e. if two functions are pointwise identifiable then they are identifiable. We want to reason with functions as it is done in standard mathematics. So, the *function extensionality axiom* is introduced to fix this problem. This axiom expresses that function:

$$\mathsf{happly}\;\mathsf{f}\;\mathsf{g}:(\mathsf{f}\sim\mathsf{g})\to(\mathsf{f}=\mathsf{g})$$

is an equivalence for any functions f and g. In other words, the axiom expresses that function $\mathsf{happly}\;\mathsf{f}\;\mathsf{g}$ has an inverse, or that the converse of Lemma 1.6.27 also holds.

- Function extensionality (S-context: Γ; S-term: f, g, A, P; Variable: w):

$$\frac{\Gamma\vdash\mathsf{f}:\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}\qquad\Gamma\vdash\mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}}{\Gamma\vdash\mathsf{funext}\;\mathsf{f}\;\mathsf{g}:\mathsf{IsEquiv}\;(\mathsf{happly}\;\mathsf{f}\;\mathsf{g})}\;\Pi\text{-EXT}$$

  Here, $(\mathsf{funext}\;\mathsf{f}\;\mathsf{g})$ is a term acting as proof to the fact that $(\mathsf{happly}\;\mathsf{f}\;\mathsf{g})$ is an equivalence. In informal proofs we will simply say "by function extensionality..." when we want to use the inverse of $(\mathsf{happly}\;\mathsf{f}\;\mathsf{g})$ without explicit mention to the Π-EXT rule.

Hence, we have the following:

Lemma 1.6.28. *Let $\mathsf{A}:\mathcal{U}_i$ be a type and $\mathsf{P}:\mathsf{A}\to\mathcal{U}_j$ a type family. The following type is inhabited:*

$$\prod_{\mathsf{f},\mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}}((\mathsf{f}=\mathsf{g})\simeq(\mathsf{f}\sim\mathsf{g}))$$

Proof. Let $\mathsf{f},\mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}$. Define:

$$\mathsf{h}_1:\equiv(\mathsf{happly}\;\mathsf{f}\;\mathsf{g}):(\mathsf{f}=\mathsf{g})\to(\mathsf{f}\sim\mathsf{g})$$
$$\mathsf{h}_2:\equiv(\mathsf{happly}\;\mathsf{f}\;\mathsf{g})^{-1}:(\mathsf{f}\sim\mathsf{g})\to(\mathsf{f}=\mathsf{g})$$

The equations:

$$\mathsf{h}_2\;(\mathsf{h}_1\;\mathsf{p})=\mathsf{p}$$
$$\mathsf{h}_1\;(\mathsf{h}_2\;\mathsf{q})=\mathsf{q}$$

are satisfied by Lemma 1.6.15, since $(\mathsf{happly}\;\mathsf{f}\;\mathsf{g})$ or h_1 is an equivalence.

□

The next property is an example of how function extensionality is used. This property expresses that functions with pairs as input can be rewritten as nested lambdas, and vice versa (these are the so-called curry and uncurry operations).

Lemma 1.6.29 (Curry and uncurry operations)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *be a type, and* $\mathsf{P}:\mathsf{A}\to\mathcal{U}_j$, $\mathsf{Q}:(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w})\to\mathcal{U}_k$ *type families.*

The function:

$$\mathsf{curry} :\equiv (\lambda \mathsf{f}.\ \lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \mathsf{f}\ (\mathsf{w},\mathsf{y})) : \left(\prod_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}} \mathsf{Q}\,\mathsf{q}\right) \to \left(\prod_{\mathsf{w}:\mathsf{A}}\prod_{\mathsf{y}:\mathsf{P}\,\mathsf{w}} \mathsf{Q}\ (\mathsf{w},\mathsf{y})\right)$$

is an equivalence (its inverse will be called the uncurry operation*).*

Hence, the following type is inhabited:

$$\left(\prod_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}} \mathsf{Q}\,\mathsf{q}\right) \simeq \left(\prod_{\mathsf{w}:\mathsf{A}}\prod_{\mathsf{y}:\mathsf{P}\,\mathsf{w}} \mathsf{Q}\ (\mathsf{w},\mathsf{y})\right)$$

Proof. To prove that curry is an equivalence, we use Lemma 1.6.16. We will use the induction principle for Σ-types to construct the inverse for curry. So, to construct:

$$\mathsf{uncurry} : \left(\prod_{\mathsf{w}:\mathsf{A}}\prod_{\mathsf{y}:\mathsf{P}\,\mathsf{w}} \mathsf{Q}\ (\mathsf{w},\mathsf{y})\right) \to \left(\prod_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}} \mathsf{Q}\,\mathsf{q}\right)$$

just define it on pairs:[40]

$$\mathsf{uncurry}\ \mathsf{f}\ (\mathsf{a},\mathsf{b}) :\equiv \mathsf{f}\ \mathsf{a}\ \mathsf{b}$$

for arbitrary $\mathsf{f}:\prod_{\mathsf{w}:\mathsf{A}}\prod_{\mathsf{y}:\mathsf{P}\,\mathsf{w}}\mathsf{Q}\ (\mathsf{w},\mathsf{y})$, and $\mathsf{a}:\mathsf{A}$, $\mathsf{b}:\mathsf{P}\,\mathsf{a}$.

Notice this equation is well-typed, because $\mathsf{f}\ \mathsf{a}\ \mathsf{b}:\mathsf{Q}\ (\mathsf{a},\mathsf{b})$.

Now, we need to check that curry and uncurry are inverses of each other.

- $\mathsf{curry}\ (\mathsf{uncurry}\ \mathsf{f}) = \mathsf{f}$ for any $\mathsf{f}:\prod_{\mathsf{w}:\mathsf{A}}\prod_{\mathsf{y}:\mathsf{P}\,\mathsf{w}}\mathsf{Q}\ (\mathsf{w},\mathsf{y})$.

  We have:

$$\begin{aligned}\mathsf{curry}\ (\mathsf{uncurry}\ \mathsf{f}) &\equiv \lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \mathsf{uncurry}\ \mathsf{f}\ (\mathsf{w},\mathsf{y})\\ &\equiv \lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \mathsf{f}\ \mathsf{w}\ \mathsf{y}\\ &\equiv \lambda \mathsf{w}.\ \mathsf{f}\ \mathsf{w}\\ &\equiv \mathsf{f}\end{aligned}$$

  where the last two steps follow by the uniqueness rule for Π-types on page 21. In more detail, given $\mathsf{w}:\mathsf{A}$, function $\mathsf{f}\ \mathsf{w}:\prod_{\mathsf{y}:\mathsf{P}\,\mathsf{w}}\mathsf{Q}\ (\mathsf{w},\mathsf{y})$ is "wrapped" inside a lambda with variable y, hence $(\lambda \mathsf{y}.\ \mathsf{f}\ \mathsf{w}\ \mathsf{y}) \equiv \mathsf{f}\ \mathsf{w}$ by the uniqueness rule.

- $\mathsf{uncurry}\ (\mathsf{curry}\ \mathsf{f}) = \mathsf{f}$ for any $\mathsf{f}:\prod_{\mathsf{q}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}}\mathsf{Q}\,\mathsf{q}$.

  Since we are trying to prove that two functions are identifiable, by the function extensionality axiom it is enough to prove that the following type is inhabited:

$$\prod_{\mathsf{p}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}} \mathsf{uncurry}\ (\mathsf{curry}\ \mathsf{f})\ \mathsf{p} = \mathsf{f}\ \mathsf{p}$$

[40] Notice we are using implicit lambda notation to introduce the parameter $\mathsf{f}:\prod_{\mathsf{w}:\mathsf{A}}\prod_{\mathsf{y}:\mathsf{P}\,\mathsf{w}}\mathsf{Q}\ (\mathsf{w},\mathsf{y})$. See Convention 1.5.10.

But, by the induction principle for Σ-types, it is enough to prove:

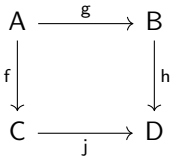
$$\prod_{\mathsf{a}:\mathsf{A}} \prod_{\mathsf{b}:\mathsf{P}\ \mathsf{a}} \mathsf{uncurry}\ (\mathsf{curry}\ \mathsf{f})\ (\mathsf{a},\mathsf{b}) = \mathsf{f}\ (\mathsf{a},\mathsf{b})$$

Hence:

$$\begin{aligned}\mathsf{uncurry}\ (\mathsf{curry}\ \mathsf{f})\ (\mathsf{a},\mathsf{b}) &\equiv \mathsf{curry}\ \mathsf{f}\ \mathsf{a}\ \mathsf{b}\\ &\equiv \mathsf{f}\ (\mathsf{a},\mathsf{b})\end{aligned}$$

□

Remark 1.6.30 (Commutative diagrams)**.** On this report, it will be common to have *commutative diagrams* involving functions. For example, given functions f, g, h, and j, whenever it is said that diagram:



commutes, this will mean that it commutes *pointwise*, i.e. type $(\mathsf{j}\circ\mathsf{f}) \sim (\mathsf{h}\circ\mathsf{g})$ is inhabited.

Because we have function extensionality, this also means that type $\mathsf{j}\circ\mathsf{f} = \mathsf{h}\circ\mathsf{g}$ is inhabited. Therefore, it makes no difference if we choose $\sim$ or $=$ as the meaning of a commutative diagram, but the pointwise version will be preferred.

▲

1.6.2 The univalence axiom

If two types are identifiable, then they are equivalent types. The following definition makes this precise.

Definition 1.6.31. We define a function:

$$\mathsf{idtoequiv}\colon \prod_{\mathsf{T}:\mathcal{U}_i} \prod_{\mathsf{R}:\mathcal{U}_j} (\mathsf{T} =_{\mathcal{U}_{\max\{i,j\}}} \mathsf{R}) \to (\mathsf{T} \simeq \mathsf{R})$$

as:

$$\mathsf{idtoequiv} :\equiv \lambda(\mathsf{T}:\mathcal{U}_i)(\mathsf{R}:\mathcal{U}_j)(\mathsf{q}:\mathsf{T}=\mathsf{R}).\ \mathsf{H}\ \mathsf{T}\ \mathsf{R}\ \mathsf{q}$$

where H is the proof of (1.15) in Lemma 1.6.22 instantiated with type:

$$\mathsf{A} :\equiv \mathcal{U}_{\max\{i,j\}}$$

and type family:

$$\mathsf{P} :\equiv \mathsf{id}_{\mathcal{U}_{\max\{i,j\}}} : \mathcal{U}_{\max\{i,j\}} \to \mathcal{U}_{\max\{i,j\}}$$

▲

The *univalence axiom* states that function $\mathsf{idtoequiv}$ is an equivalence, i.e. it has an inverse, which expresses that equivalent types can be identified.

- Univalence (S-context: Γ; S-term: T, R):

$$\frac{\Gamma \vdash \mathsf{T}:\mathcal{U}_i \qquad \Gamma \vdash \mathsf{R}:\mathcal{U}_j}{\Gamma \vdash \mathsf{univalence}\ \mathsf{T}\ \mathsf{R}:\mathsf{IsEquiv}\ (\mathsf{idtoequiv}\ \mathsf{T}\ \mathsf{R})}\ \mathcal{U}_{\max\{i,j\}}\text{-UNIV}$$

  Here, $(\mathsf{univalence}\ \mathsf{T}\ \mathsf{R})$ is a term acting as proof to the fact that $(\mathsf{idtoequiv}\ \mathsf{T}\ \mathsf{R})$ is an equivalence. In informal proofs we will simply say "by univalence..." when we want to use the inverse of $(\mathsf{idtoequiv}\ \mathsf{T}\ \mathsf{R})$ without explicit mention to the $\mathcal{U}_{\max\{i,j\}}$-UNIV rule.

Hence, we have the following result.

Lemma 1.6.32. *The following type is inhabited:*

$$\prod_{\mathsf{T}:\mathcal{U}_i} \prod_{\mathsf{R}:\mathcal{U}_j} (\mathsf{T} =_{\mathcal{U}_{\max\{i,j\}}} \mathsf{R}) \simeq (\mathsf{T} \simeq \mathsf{R})$$

Proof. Let $\mathsf{T}:\mathcal{U}_i$ and $\mathsf{R}:\mathcal{U}_j$. Define:

$$\mathsf{h}_1 :\equiv (\mathsf{idtoequiv}\ \mathsf{T}\ \mathsf{R}):(\mathsf{T}=\mathsf{R}) \to (\mathsf{T}\simeq\mathsf{R})$$
$$\mathsf{h}_2 :\equiv (\mathsf{idtoequiv}\ \mathsf{T}\ \mathsf{R})^{-1}:(\mathsf{T}\simeq\mathsf{R}) \to (\mathsf{T}=\mathsf{R})$$

The equations:

$$\mathsf{h}_2\ (\mathsf{h}_1\ \mathsf{p}) = \mathsf{p}$$
$$\mathsf{h}_1\ (\mathsf{h}_2\ \mathsf{q}) = \mathsf{q}$$

are satisfied by Lemma 1.6.15, since $(\mathsf{idtoequiv}\ \mathsf{T}\ \mathsf{R})$ or h_1 is an equivalence.

□

One interesting consequence of univalence is that it is not necessary to add function extensionality as an additional axiom to HoTT (as we did in Section 1.6.1), because function extensionality logically follows from univalence (see Section 4.9 in the HoTT book [20] for a proof of this).

If we interpret types as sets and the identity type as equality, the univalence axiom then expresses that isomorphic sets are equal, which is clearly nonsensical. This implies that, in HoTT, statements of the form $\mathsf{w}=\mathsf{y}$ should *not* be interpreted as expressing equality between w and y in the sense of standard mathematics, but only that w and y are *identifiable* (by some "mechanism", if you like), or that they are *indistinguishable* from the point of view of the theory. The word "identifiable" should suggest a weaker notion than the standard mathematical notion of equality. Section 1.7 will explore further what we mean by "identifiable".

In fact, in standard mathematics, it is a common *informal* practice to "identify" isomorphic structures. The reason is that under a certain mathematical domain, theorems are usually invariant under isomorphism. The univalence axiom formally justifies this practice by claiming that equivalent types (i.e. isomorphic types) can be identified. In other words, in HoTT, *all theorems will be invariant under equivalence.* For an example on how univalence makes possible the identification of isomorphic structures, see Section 2.14 in the HoTT book, where it is shown that two semigroups can be identified precisely when they are isomorphic in the algebraic sense.

However, the main use of univalence is to transfer results proved for some type A into an equivalent type B. For example, let us suppose we proved $\mathsf{t}:\mathsf{Q}\ \mathsf{A}$ which expresses that property Q holds for type A. Since B is equivalent to A, by univalence we will have a proof $\mathsf{q}:\mathsf{A}=\mathsf{B}$, and hence $\mathsf{transport}^{\mathsf{Q}}\ \mathsf{q}\ \mathsf{t}:\mathsf{Q}\ \mathsf{B}$. In other words, we will have a proof that Q also holds for type B.

1.7 The homotopy interpretation

For this section, the author assumes that the reader is already familiar with basic concepts of algebraic topology. The reader is invited to read Hatcher's book [12]. A reference book is also Aguilar, Gitler & Prieto [2].

We have not explained why HoTT has the word "homotopy" on its name. HoTT is Martin Löf's Intentional Type Theory extended with the univalence axiom and higher inductive types [20] (higher inductive types will be presented in Section 1.11).

It is known that Martin Löf's Intentional Type Theory has topological models ([23], [6], and [21]). In particular, when the univalence axiom is added, HoTT has a model (or semantics) in simplicial sets [14], which have CW complexes (a special kind of topological space) as their realization [10].

In the topological semantics for HoTT, homotopies (as understood in algebraic topology) play an important role. Hence the word "homotopy" on the type theory name.

Although the semantics for HoTT make use of CW complexes (more specifically, simplicial sets), when the semantics are explained informally, it is easier to think in terms of general topological spaces instead of simplicial sets and CW complexes. This is what we will do here.

Type theory	***Homotopy***	***Logical***
A is a type	A is a topological space	A is a proposition
$\tau : \mathsf{A}$	τ is a point in space A	τ is a proof for proposition A
$\mathsf{A} \to \mathsf{B}$ (non-dependent functions)	Space of continuous functions from A to B	Proofs for implication: functions mapping proofs for A into proofs for B
$\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ (dependent pairs)	Total space for fibration $\mathsf{pr}_1 : (\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}) \to \mathsf{A}$	Proofs for existential: a proof w for A together with a proof that w satisfies P
$\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ (dependent functions)	Space of continuous sections for fibration $\mathsf{pr}_1 : (\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}) \to \mathsf{A}$	Proofs for universal statement: functions mapping a proof w for A into a proof that w satisfies P
$\mathsf{A} \times \mathsf{B}$ (non-dependent pairs)	Product space	Proofs for conjunction: a proof for A together with a proof for B
$\mathsf{A} + \mathsf{B}$ (case injections)	Disjoint union of spaces	Proofs for disjunction: either a proof for A or a proof for B, indicating which case holds
$\mathbb{0}$ (empty type)	Empty space	Proofs for contradiction
$\mathbb{1}$ (singleton type)	Singleton space	Proofs for tautology
$\mathsf{w} =_{\mathsf{A}} \mathsf{y}$ (identifications)	Space of paths in A starting at point w and ending at point y	Proofs for the proposition claiming that proofs w and y are identifiable
$\mathsf{refl}_{\mathsf{A}}\ \mathsf{w}$ (trivial proof for $\mathsf{w} = \mathsf{w}$)	Constant loop at point w	Reflexivity proof (or trivial proof) for $\mathsf{w} = \mathsf{w}$

Table 1.1: Interpretations for HoTT

This section will give only an informal overview of the topological semantics for HoTT, since a full formal description will not be required. The reason is that the topological semantics will not play a role on this report, as all the results on this document can be understood by working entirely within the intended type theoretic meaning (as described in all previous sections), mixed with the logical interpretation. If we were doing homotopy theory inside HoTT, then the topological semantics would be critical for our development. See for example Chapter 8 in the HoTT Book [20].

However, it would be odd to present HoTT without explaining its most awe-inspiring aspect: HoTT has a topological interpretation. Also, the topological semantics serve to clarify the logical meaning of identity types, as we will do at the end of this section.

To motivate the homotopy interpretation (or topological interpretation or topological model) for HoTT, let us have a look on a type that cannot be proved to be inhabited in HoTT:

$$\prod_{\mathsf{A}:\mathcal{U}_i} \prod_{\mathsf{w}:\mathsf{A}} \prod_{\mathsf{p}:\mathsf{w}=\mathsf{w}} \mathsf{p} = \mathsf{refl}_{\mathsf{A}}\ \mathsf{w} \tag{1.19}$$

If we attempt to prove (1.19) by an appeal to the induction principle for identity types (Lemma 1.5.57), we soon realize that it is not clear how to apply the principle, because type $\mathsf{w} = \mathsf{w}$ has variable w fixed on both sides. Remember that the induction principle proves statements of the form $\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} \prod_{\mathsf{p}:\mathsf{w}=\mathsf{y}} \mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p}$, where type $\mathsf{w} = \mathsf{y}$ does *not* have variable w fixed on both sides.

Type (1.19) expresses that any proof for $\mathsf{w} = \mathsf{w}$ (for an arbitrary $\mathsf{w} : \mathsf{A}$) must be the trivial proof $\mathsf{refl}_{\mathsf{A}}\ \mathsf{w}$. It is reasonable to think that this type should not be inhabited since we have function extensionality and univalence, which are ways for introducing identity proofs that are different from the trivial proof.

However, if we remove function extensionality and univalence, this type still cannot be proved to be inhabited.[41] This seems surprising because univalence and function extensionality seem to be the only way to produce identity proofs different from the trivial proof.

Also, why is the elimination rule for identity types still valid if trivial proofs are not the only available proofs?

The homotopy interpretation provides an answer to these questions. Table 1.1 summarizes the homotopy interpretation in the "homotopy" column. We also add the logical interpretation for comparison (in the "logical" column). We proceed to explain the homotopy interpretation.

[41] Because the topological model is still a model if univalence and function extensionality are removed from HoTT.

The most surprising aspect of the homotopy interpretation is the meaning assigned to identity types. Given $\mathsf{w}, \mathsf{y} : \mathsf{A}$, type $\mathsf{w} =_{\mathsf{A}} \mathsf{y}$ denotes the space (in standard mathematical notation):

$$\{p : I \to A \mid p(0) = w,\ p(1) = y\}$$

of *paths* starting at point w and ending at point y. Here, I denotes the unit interval $[0, 1] \subseteq \mathbb{R}$.

So, what is the meaning of type $\mathsf{p} =_{(\mathsf{w}=\mathsf{y})} \mathsf{q}$? It should be the space of paths:

$$\{H : I \to \{r : I \to A \mid r(0) = w,\ r(1) = y\} \mid H(0) = p,\ H(1) = q\}$$

which, by the exponential law,[42] can be rewritten as:

$$\{H : I \times I \to A \mid \forall t \in I.\ H(0, t) = p(t),\ H(1, t) = q(t),\ H(t, 0) = w,\ H(t, 1) = y\}$$

which corresponds to the space of all homotopies rel end points between paths p and q. In other words, any term of type $\mathsf{p} =_{(\mathsf{w}=\mathsf{y})} \mathsf{q}$ can be interpreted as an homotopy rel end points between paths p and q.

If we continue this analysis, any term of type $\mathsf{H} =_{(\mathsf{p}=_{(\mathsf{w}=\mathsf{y})}\mathsf{q})} \mathsf{Q}$ will be a 2-homotopy between homotopies H and Q (i.e. an "homotopy between surfaces H and Q" as homotopies can be seen as "surfaces"), and so on.

Another interesting aspect of the homotopy interpretation is the meaning given to types $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ and $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$.

First, terms of type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ can be seen as sections for the projection function $\mathsf{pr}_1 : (\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}) \to \mathsf{A}$ (ignore for the moment that Table 1.1 claims that pr_1 is a fibration). Given $\mathsf{f} : \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$, we can form the *graph function* for f as:

$$\mathsf{graph}_{\mathsf{f}} :\equiv (\lambda \mathsf{w}.\ (\mathsf{w}, \mathsf{f}\ \mathsf{w})) : \mathsf{A} \to \left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right)$$

which *is* a section for pr_1:

$$\mathsf{pr}_1\ (\mathsf{graph}_{\mathsf{f}}\ \mathsf{a}) \equiv \mathsf{pr}_1\ (\mathsf{a}, \mathsf{f}\ \mathsf{a}) \equiv \mathsf{a}$$

Also, under the homotopy interpretation, any function is *continuous*. This means that functions of type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ are *continuous sections* for pr_1.

Now, type $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ can be seen as the total space for the fibration $\mathsf{pr}_1 : (\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}) \to \mathsf{A}$. But, what do we mean by a *fibration*?

In algebraic topology, a *fibration* [2] (or *Hurewicz fibration*) is a continuous function $p : E \to B$ with the *homotopy lifting property*, that is: if X is a topological space, $f : X \to E$ a continuous function, and $H : X \times I \to B$ an homotopy such that the following square commutes:

$$\begin{array}{ccc} X & \xrightarrow{f} & E \\ {\scriptstyle j_0}\downarrow & & \downarrow{\scriptstyle p} \\ X \times I & \xrightarrow[H]{} & B \end{array}$$

where j_0 is function $j_0(x) = (x, 0)$, then, there is an homotopy $H' : X \times I \to E$ making the two triangles commute:

$$\begin{array}{ccc} X & \xrightarrow{f} & E \\ {\scriptstyle j_0}\downarrow & \overset{H'}{\nearrow} & \downarrow{\scriptstyle p} \\ X \times I & \xrightarrow[H]{} & B \end{array}$$

Here, E is called the *total space* for the fibration, B is called the *base space* for the fibration, H' is called the *lifted homotopy*, and for every $b \in B$, $p^{-1}(\{b\})$ is called the *fiber* for b.

However, there is an alternative characterization for fibrations involving a path-lifting map,[43] which we explain next.

[42] See Section 1.3 in Aguilar, et al. [2].
[43] See Section 4.3 in Aguilar, et al. [2].

First, given a continuous function $p : E \to B$, define the space:

$$E \times_B B^I :\equiv \{(e, \alpha) \in E \times B^I \mid p(e) = \alpha(0)\}$$

where B^I denotes the space of paths in B, i.e. B^I is the space of all continuous functions $I \to B$.

With this definition, a continuous function $p : E \to B$ is a *fibration* if and only if there is a continuous function:

$$\Gamma : E \times_B B^I \to E^I$$

such that for every $(e, \alpha) \in E \times_B B^I$, we have $\Gamma(e, \alpha)(0) = e$ and $p(\Gamma(e, \alpha)(t)) = \alpha(t)$ for every $t \in I$.

In other words, Γ is a function that takes as input:

- A "starting" point e in the total space E.
- A path α in the base space B. This path must start at a point "below" e, i.e. condition "$p(e) = \alpha(0)$" in the definition of $E \times_B B^I$.

and produces as output:

- A path in the total space. This path will start at e (i.e. condition "$\Gamma(e, \alpha)(0) = e$"), and it will be "above" α (i.e. condition "$p(\Gamma(e, \alpha)(t)) = \alpha(t)$ for every $t \in I$").

Hence, Γ *continuously* "lifts" paths on the base space into the total space.

With this characterization, we can now check that $\mathsf{pr_1} : (\sum_{\mathsf{w:A}} \mathsf{P\ w}) \to \mathsf{A}$ is a fibration. For this, we need to construct a continuous path-lifting function that lifts paths from the base space A into the total space $\sum_{\mathsf{w:A}} \mathsf{P\ w}$.

This is achieved by proving that the following type is inhabited:

$$\prod_{\mathsf{u:A}} \prod_{\mathsf{z:P\ u}} \prod_{\mathsf{y:A}} \prod_{\alpha:\mathsf{u=y}} \left(\sum_{\beta : (\mathsf{u},\ \mathsf{z})=(\mathsf{y},\ \alpha_*\ \mathsf{z})} \mathsf{ap}_{\mathsf{pr_1}}\ \beta = \alpha \right) \tag{1.20}$$

where $\mathsf{A} : \mathcal{U}_i$ is a given type and $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$ a given type family.

Let us dissect this type. To prove this type is inhabited, we have to construct a function Γ that will receive as input:

- A point $\mathsf{u} : \mathsf{A}$ in the base space, and a point $\mathsf{z} : \mathsf{P\ u}$. Equivalently, we are given a point $(\mathsf{u}, \mathsf{z}) : \sum_{\mathsf{w:A}} \mathsf{P\ w}$ in the total space. The pair (u, z) is analogous to point $e \in E$ in the characterization.
- A path[44] $\alpha : \mathsf{u} = \mathsf{y}$ in the base space A. Path α starts at point $\mathsf{pr_1}\ (\mathsf{u}, \mathsf{z}) \equiv \mathsf{u}$, which is "below" point (u, z). This is analogous to condition "$p(e) = \alpha(0)$" in the characterization.

and will produce as output:

- A path $\beta : (\mathsf{u},\ \mathsf{z}) = (\mathsf{y},\ \alpha_*\ \mathsf{z})$ in the total space, i.e. the lifted path. This path starts at point (u, z). This is analogous to condition "$\Gamma(e, \alpha)(0) = e$" in the characterization.

  Since we have to specify where our path ends, a natural point to choose is one "above" y. Notice that we have the transport function $\alpha_* : (\mathsf{P\ u}) \to (\mathsf{P\ y})$. Therefore, it is reasonable to choose point $(\mathsf{y}, \alpha_*\ \mathsf{z})$ as the end point, as $(\mathsf{y}, \alpha_*\ \mathsf{z})$ is "above" y, i.e. $\mathsf{pr_1}\ (\mathsf{y}, \alpha_*\ \mathsf{z}) \equiv \mathsf{y}$.

- An homotopy $\mathsf{H} : \mathsf{ap}_{\mathsf{pr_1}}\ \beta = \alpha$ between paths:

$$\mathsf{ap}_{\mathsf{pr_1}}\ \beta : \mathsf{pr_1}\ (\mathsf{u},\ \mathsf{z}) = \mathsf{pr_1}\ (\mathsf{y},\ \alpha_*\ \mathsf{z})$$
$$\alpha : \mathsf{u} = \mathsf{y}$$

  This is analogous to condition: "$p(\Gamma(e, \alpha)(t)) = \alpha(t)$ for every $t \in I$" in the characterization.

  However, condition $p(\Gamma(e, \alpha)(t)) = \alpha(t)$ is a *strict* equality, while HoTT can only express that $\mathsf{ap}_{\mathsf{pr_1}}\ \beta$ is *homotopic rel end points* to α. HoTT is unable to express that two paths are exactly the same. It can only express that paths are equal up to homotopy.

[44] Notice we have to introduce the parameter $\mathsf{y} : \mathsf{A}$ in type (1.20) to express that path α ends at *some* point. In HoTT, we always have to specify the point where a path ends.

The proof that type (1.20) is inhabited is quite trivial. It follows by one application of the induction principle for identity types (Lemma 1.5.57). We omit the proof as this is not essential for our development.

Since $\sum_{\mathsf{w:A}} \mathsf{P\ w}$ is the total space for fibration $\mathsf{pr_1}$, type $\mathsf{P\ w}$ (for $\mathsf{w:A}$) can be interpreted as the *fiber* for w.

With the homotopy interpretation at hand, we can explain why type (1.19) cannot be inhabited. Type (1.19) is claiming that there is a continuous function Δ such that given an arbitrary space A, a point w on A, and a loop p on w, function Δ will produce as output an homotopy rel end points between loop p and the constant loop at w.

This function is impossible to construct, for take the circle $(S^1, *)$, where $*$ is its base point. We know that there is no homotopy rel end points between the circle S^1 and the constant loop at $*$, for otherwise, $(S^1, *)$ will have a trivial fundamental group, which is not the case. Therefore, if function Δ exists, we will be able to construct an homotopy between S^1 and the constant loop at $*$, which is a contradiction.

The homotopy interpretation provides quite interesting meanings to the results on previous sections. For example, the symmetry operator (Lemma 1.5.59) can be interpreted as a *path inversion* operator (i.e. the path that results by traversing the original path "in reverse"), and the transitivity operator (Lemma 1.5.60) can be interpreted as a *path concatenation* operator.

Hence, under the homotopy interpretation, Lemma 1.6.5 expresses the usual properties on paths known in homotopy theory:

- If we concatenate a path p with a constant loop, we will get a path that is homotopic to the original path p.
- If we concatenate a path p and its inverse path p^{-1}, we will get a path that is homotopic to a constant loop.
- If we invert a path p twice, we will get a path that is homotopic to the original path p.

Also, Definition 1.6.7 expresses that type $\mathsf{f} \sim \mathsf{g}$ (for $\mathsf{f,g:} \prod_{\mathsf{w:A}} \mathsf{P\ w}$) is inhabited when there is a continuous function assigning to each point $w \in A$, a path from $f(w)$ to $g(w)$. By the exponential law, this is equivalent to claiming that there is an homotopy H between functions f and g. In other words, type $\mathsf{f} \sim \mathsf{g}$ means that continuous functions f and g are homotopic.

Therefore, it follows that type $\mathsf{A} \simeq \mathsf{B}$ (Definition 1.6.10) means that spaces A and B are homotopy equivalent.

As another example, Lemma 1.6.22 expresses a well known property for fibrations: if there is a path p connecting points w and y in the base space, then the fibers of w and y will be homotopy equivalent spaces.

The function extensionality axiom (see Section 1.6.1) expresses that paths in function spaces correspond to homotopies between functions.

The univalence axiom (see Section 1.6.2) expresses that paths in universes (a universe is a space of spaces) correspond to homotopy equivalences between spaces. So, in the homotopy interpretation, univalence makes sense because $(=)$ denotes paths and *not* strict equality.

Since $(=)$ denotes *paths in a space*, our use of the word "identifiable" when working inside the logical interpretation, can be explained as follows.

Given a proof $\mathsf{p:w = y}$ that w and y are identifiable, p can be interpreted as a "procedure or algorithm" that performs the identification of w and y. In the homotopy interpretation, between any two different points $w, y \in A$, there may be many different paths connecting them. Therefore, when this is translated into the logical interpretation, it must be the case that between any two non-definitionally equal terms, there may be many different procedures (or proofs) identifying them.[45] Therefore, $\mathsf{w = y}$ cannot mean that w and y are identical in the sense of standard mathematics, because w and y may be non-definitionally equal terms. It is only required the existence of a proof (or procedure) that performs the identification between w and y to be able to conclude that those two terms are *indistinguishable* from the point of view of the theory. To summarize, "identifiable" means *equal modulo a proof or procedure that performs the identification.*

Under this remark, univalence does make sense, because it claims that types are *identifiable* if they are equivalent. It does *not* claim that types are exactly the same if they are equivalent. Univalence simply claims that there is always a way to construct a procedure that performs the identification when we already know that there is an equivalence between the two types.

[45] Because a path in the homotopy interpretation corresponds to an identification procedure or proof in the logical interpretation.

We are ready to explain why the elimination rule for identity types is sound even under the presence of non-trivial proofs for $\mathsf{w} = \mathsf{w}$ (i.e. non-constant loops at w).

To make the explanation easier, we will work with the induction principle for identity types (Lemma 1.5.57), which we will call "IP", instead of working with the elimination and computation rules. The reason is that IP is "more natural" to work with, as it states everything in terms of functions and type families, while the inference rules make excessive use of substitutions and judgments. In spite of working with IP, our explanation will still apply to the elimination rule, because IP is just the elimination and computation rules "in disguise", i.e. IP is provable from the elimination and computation rules (as we did in Lemma 1.5.57), and conversely, the elimination and computation rules are derivable from IP (though we omit the proof of this, as it is not critical for the developments on this report).

The explanation will be based on the following lemma.

Lemma 1.7.1. *Let* **(TR)** *and* **(UP)** *denote the following statements.*

(TR) *There is a function:*

$$\mathsf{T} : \prod_{\mathsf{A}:\mathcal{U}_i} \prod_{\mathsf{C}:\mathsf{A}\to\mathcal{U}_j} \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} (\mathsf{w} = \mathsf{y}) \to (\mathsf{C}\ \mathsf{w}) \to (\mathsf{C}\ \mathsf{y})$$

such that for any $\mathsf{A} : \mathcal{U}_i$, $\mathsf{C} : \mathsf{A} \to \mathcal{U}_j$, *and* $\mathsf{a} : \mathsf{A}$, *it satisfies:*

$$\mathsf{T}\ \mathsf{A}\ \mathsf{C}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{id}_{(\mathsf{C}\ \mathsf{a})} \tag{1.21}$$

In other words, T *is a "transport operator" (notice the similarity with Lemma 1.5.61).*

Parameters A, w, *and* y *will be left implicit, so that Equation (1.21) can be written as:*

$$\mathsf{T}\ \mathsf{C}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{id}_{(\mathsf{C}\ \mathsf{a})} \tag{1.22}$$

(UP) *There is a function:*

$$\mathsf{U} : \prod_{\mathsf{A}:\mathcal{U}_i} \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} \prod_{\mathsf{p}:\mathsf{w}=\mathsf{y}} (\mathsf{w},\ (\mathsf{w},\ \mathsf{refl}_{\mathsf{A}}\ \mathsf{w})\ ) = (\ \mathsf{w},\ (\mathsf{y},\ \mathsf{p})\ )$$

such that for any $\mathsf{A} : \mathcal{U}_i$ *and* $\mathsf{a} : \mathsf{A}$, *it satisfies:*

$$\mathsf{U}\ \mathsf{A}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) \equiv \mathsf{refl}_{\left(\sum_{\mathsf{w}:\mathsf{A}} \sum_{\mathsf{y}:\mathsf{A}} \mathsf{w}=\mathsf{y}\right)}\ (\ \mathsf{a},\ (\mathsf{a},\ \mathsf{refl}_{\mathsf{A}}\ \mathsf{a})\ ) \tag{1.23}$$

Function U *acts as a proof for the so-called "Uniqueness principle for identity types".*

Then, the following implications hold.

(i) **(TR)** *and* **(UP)** *together imply* IP.

(ii) IP *implies both* **(TR)** *and* **(UP)**.

Proof. We prove each case as follows.

(i). Suppose functions T and U. Since we want to prove IP, we suppose the following:

$$\mathsf{A} : \mathcal{U}_i$$
$$\mathsf{C} : \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} ((\mathsf{w} = \mathsf{y}) \to \mathcal{U}_j)$$
$$\mathsf{g} : \prod_{\mathsf{a}:\mathsf{A}} \mathsf{C}\ \mathsf{a}\ \mathsf{a}\ (\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})$$

and then construct the required function f.

First, we define a type family $\mathsf{D} : (\sum_{\mathsf{w}:\mathsf{A}} \sum_{\mathsf{y}:\mathsf{A}} \mathsf{w} = \mathsf{y}) \to \mathcal{U}_j$ by the recursion principle for Σ-types (applied twice) as:

$$\mathsf{D}\ (\ \mathsf{w},\ (\mathsf{y},\ \mathsf{p})\ ) :\equiv \mathsf{C}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p}$$

Now, define function f as:

$$\mathsf{f\ w\ y\ p} :\equiv \mathsf{T\ D\ (U\ A\ w\ y\ p)\ (g\ w)}$$

Notice f is well-typed, since:

$$\mathsf{g\ w} : \mathsf{C\ w\ w\ (refl_A\ w)}$$

or equivalently, by definition of family D:

$$\mathsf{g\ w} : \mathsf{D\ (\ w,\ (w,\ refl_A\ w)\ )}$$

Similarly:

$$\mathsf{T\ D\ (U\ A\ w\ y\ p)\ (g\ w)} : \mathsf{D\ (\ w,\ (y,\ p)\ )}$$

or equivalently, by definition of D:

$$\mathsf{T\ D\ (U\ A\ w\ y\ p)\ (g\ w)} : \mathsf{C\ w\ y\ p}$$

Finally, for any $\mathsf{a} : \mathsf{A}$, we have:

$$\begin{aligned}
\mathsf{f\ a\ a\ (refl_A\ a)} &\equiv \mathsf{T\ D\ (U\ A\ a\ a\ (refl_A\ a))\ (g\ a)} \\
&\overset{(1)}{\equiv} \mathsf{T\ D\ (refl}_{\left(\sum_{w:A}\sum_{y:A} w=y\right)}\ \mathsf{(\ a,\ (a,\ refl_A\ a)\ ))\ (g\ a)} \\
&\overset{(2)}{\equiv} \mathsf{id}_{(D\ (\ a,\ (a,\ refl_A\ a)\ ))}\ \mathsf{(g\ a)} \\
&\equiv \mathsf{id}_{(C\ a\ a\ (refl_A\ a))}\ \mathsf{(g\ a)} \\
&\equiv \mathsf{g\ a}
\end{aligned}$$

where (1) follows by (1.23), and (2) by (1.22).

(ii). Suppose IP.
(TR) follows by a proof similar to the one for the transport operator in Lemma 1.5.61.
(UP) follows by IP, since Equation (1.23) defines function U on trivial proofs, as required by IP.

□

Lemma 1.7.1 "reveals" the intuition behind IP (or the elimination and computation rules), because IP is *really clamming two things*:

- **(TR)** claims that, for all involved spaces in our theory, it is always possible to continuously transport fibers along a path in any base space.

  Observe that in the continuous path-lifting map (1.20), the transport function is used to obtain the endpoint of the lifted path. Therefore, **(TR)** is also expressing that, for all involved spaces in our theory, we can continuously obtain the point where lifted paths will end, as long as we provide a starting point for the lifted path.

- **(UP)** claims something very interesting. Paths of the form:

  $$\mathsf{(w,\ (w,\ refl_A\ w)\ )} = \mathsf{(\ w,\ (y,\ p)\ )}$$

  correspond to homotopies between the constant loop $\mathsf{refl_A\ w}$ and path p, where *the endpoints on both paths can freely move during the homotopy.*[46]

  Therefore, **(UP)** claims that, for all involved spaces in our theory, it is always possible to continuously assign homotopies (with free endpoints) between constant loops $\mathsf{refl_A\ w}$ and paths starting at w.

[46] See Section 2.7 and Lemma 2.11.2 in the HoTT book [20].

This claim is sensible in standard mathematics, because we can always construct an homotopy (with free endpoints) between a constant loop at $w \in A$:

$$c : I \to A$$
$$c_w(x) = w$$

and a path between points $w, y \in A$:

$$p_{w,y} \in \{q : I \to A \mid q(0) = w \text{ and } q(1) = y\}$$

as follows:

$$H : I \times I \to A$$
$$H(r, t) = p_{w,y}(rt)$$

since $H(r, 0) = p_{w,y}(0) = w = c_w(r)$ for any $r \in I$, and $H(r, 1) = p_{w,y}(r)$ for any $r \in I$.

Let us summarize the claims made by IP. At the moment we introduce into HoTT the elimination and computation rules for identity types, we are implicitly introducing the claims that fibers connected through a path can be transported along it, *and* that any constant loop at some point can be deformed by an homotopy with free endpoints into a path starting at the point.

Hence, the elimination and computation rules are sound not because ($\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}$) are the only proofs for identity types (which is false), but because constant loops can be deformed into non-constant paths by homotopies with free endpoints. In the logical interpretation this means that trivial proofs can be transformed into non-trivial proofs, and the elimination rule is claiming that there *is* a procedure doing this.

Although we have barely scratched the surface of the homotopy interpretation for HoTT, this section should suggest to the reader that HoTT is a very interesting language for doing homotopy theory. For example, many spaces can be defined in HoTT and get their homotopy groups investigated. Another advantage is that any function definable between these spaces will be automatically continuous. For further details on the homotopy interpretation, the reader is invited to have a look at Chapters 2, 6, 7, and 8 in the HoTT book [20].

1.8 Contractible types, propositions and sets

A contractible type will be a type with only one element up to identification.

Definition 1.8.1 (Contractible type)**.** Let $\mathsf{A} : \mathcal{U}_i$ be a type. We define type:

$$\mathsf{IsContr}\ \mathsf{A} :\equiv \sum_{\mathsf{a}:\mathsf{A}} \prod_{\mathsf{w}:\mathsf{A}} \mathsf{a} = \mathsf{w}$$

If type $\mathsf{IsContr}\ \mathsf{A}$ is inhabited, we say that A is a *contractible type*. Contractible types are also called (-2)-*types*. We call term $\mathsf{a} : \mathsf{A}$ the *center of contraction* for A, and denote it as $\mathsf{center}\ \mathsf{A}$.

▲

Hence, A is contractible if every term in A can be identified with its center of contraction (i.e. A has exactly one element up to identification).

The name "contractible" comes from the homotopy interpretation. Type $\mathsf{IsContr}\ \mathsf{A}$ states that A is a space with a point $a \in A$ and a continuous function $H : A \to A^I$ that assigns to each $w \in A$ a path $p \in A^I$ such that $p(0) = a$ and $p(1) = w$. Since H is continuous, by the exponential law we get a continuous function $H' : A \times I \to A$ such that $H'(w, 0) = a$ and $H'(w, 1) = w$. This H' corresponds to a homotopy between the constant function at a and the identity function on A. Therefore, the space A is contractible.

An example of contractible type is $\mathbb{1}$, because all its terms are identifiable with $\star$ by Lemma 1.6.2. Here, $\star$ acts as the center of contraction for $\mathbb{1}$.

Another example is provided by the following lemma.

Lemma 1.8.2. *Let* $\mathsf{D} : \mathcal{U}_i$ *be a type. Then, the following type is inhabited:*

$$\mathsf{IsContr}\ (\mathbb{0} \to \mathsf{D})$$

i.e. there is only one function with domain $\mathbb{0}$ *and codomain an arbitrary type* D*.*

Proof. By the recursion principle for $\mathbb{0}$ (Lemma 1.5.42), there is a function $\mathsf{f} : \mathbb{0} \to \mathsf{D}$. This f will be the center of contraction.

Now, given another $\mathsf{g} : \mathbb{0} \to \mathsf{D}$, we have to prove $\mathsf{f} = \mathsf{g}$. By function extensionality, it is enough to prove:

$$\prod_{\mathsf{w}:\mathbb{0}} \mathsf{f}\ \mathsf{w} = \mathsf{g}\ \mathsf{w}$$

but this is trivially inhabited by the induction principle for $\mathbb{0}$ (Lemma 1.5.41).

□

The next property provides a characterization for Σ-types when the type family $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$ is a family of contractible types.

Lemma 1.8.3. *Let* $\mathsf{A} : \mathcal{U}_i$ *and* $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$*. If for every* $\mathsf{w} : \mathsf{A}$*, type* $\mathsf{P}\ \mathsf{w}$ *is contractible, then the following type is inhabited:*

$$\left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right) \simeq \mathsf{A}$$

Proof. We need to prove a type equivalence, so we need to define two mutually inverse functions between the types (Lemma 1.6.18).

The first function will be the projection function:

$$\mathsf{pr}_1 : \left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right) \to \mathsf{A}$$

The second function is defined as follows:

$$\mathsf{f} :\equiv (\lambda \mathsf{w}:\mathsf{A}.\ (\mathsf{w}, \mathsf{center}\ (\mathsf{P}\ \mathsf{w}))) : \mathsf{A} \to \left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right)$$

For $\mathsf{w} : \mathsf{A}$, equation $\mathsf{pr}_1\ (\mathsf{f}\ \mathsf{w}) = \mathsf{w}$ follows:

$$\mathsf{pr}_1\ (\mathsf{f}\ \mathsf{w}) \equiv \mathsf{pr}_1\ (\mathsf{w}, \mathsf{center}\ (\mathsf{P}\ \mathsf{w})) \equiv \mathsf{w}$$

For the other equation $\mathsf{f}\ (\mathsf{pr}_1\ \mathsf{w}) = \mathsf{w}$, by the induction principle for Σ-types it is enough to prove:

$$\mathsf{f}\ (\mathsf{pr}_1\ (\mathsf{a}, \mathsf{b})) = (\mathsf{a}, \mathsf{b})$$

for any $\mathsf{a} : \mathsf{A}$ and $\mathsf{b} : \mathsf{P}\ \mathsf{a}$.

We have by definition:

$$\mathsf{f}\ (\mathsf{pr}_1\ (\mathsf{a}, \mathsf{b})) \equiv \mathsf{f}\ \mathsf{a} \equiv (\mathsf{a}, \mathsf{center}\ (\mathsf{P}\ \mathsf{a}))$$

So, it remains to prove $(\mathsf{a}, \mathsf{center}\ (\mathsf{P}\ \mathsf{a})) = (\mathsf{a}, \mathsf{b})$. By Lemma 1.6.25 it is enough to construct a proof $\mathsf{q} : \mathsf{a} = \mathsf{a}$ such that $\mathsf{q}_*\ (\mathsf{center}\ (\mathsf{P}\ \mathsf{a})) = \mathsf{b}$ is inhabited.

So, consider $\mathsf{refl}_{\mathsf{A}}\ \mathsf{a} : \mathsf{a} = \mathsf{a}$. Type $(\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})_*\ (\mathsf{center}\ (\mathsf{P}\ \mathsf{a})) = \mathsf{b}$ simplifies to $\mathsf{center}\ (\mathsf{P}\ \mathsf{a}) = \mathsf{b}$ which is inhabited since $\mathsf{P}\ \mathsf{a}$ is contractible (i.e. any term $\mathsf{b} : \mathsf{P}\ \mathsf{a}$ will be identifiable with the center of $\mathsf{P}\ \mathsf{a}$).

□

Now, we introduce the notion of mere proposition.

Definition 1.8.4 (Mere proposition)**.** Let $\mathsf{A} : \mathcal{U}_i$ be a type. We define type:

$$\mathsf{IsProp}\ \mathsf{A} :\equiv \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} \mathsf{w} = \mathsf{y}$$

If type $\mathsf{IsProp}\ \mathsf{A}$ is inhabited, we say that A is a *mere proposition* or simply a *proposition*. Mere propositions are also called (-1)-*types*.

We also define type:

$$\mathbf{Prop}_i :\equiv \sum_{\mathsf{A}:\mathcal{U}_i} \mathsf{IsProp}\ \mathsf{A}$$

called the *universe of mere propositions* at level i.

▲

A mere proposition is a type where any two of its terms can be identified. In other words, under the Curry-Howard correspondence, a mere proposition is a logical statement where any of its proofs is as good as any other (because all its proofs are identifiable). Mere propositions are *proof-irrelevant*.

Notice we have now two uses of the word "proposition". When we want to interpret a type as a proposition according to the Curry-Howard correspondence, we will explicitly say so. Instead, if we just say that a type is a proposition, this will mean that it satisfies Definition 1.8.4.

Under the homotopy interpretation, a mere proposition is a path-connected space that contracts to a point if the space is not empty, i.e. *if* there is a point $\mathsf{a} : \mathsf{A}$, then we have $\prod_{\mathsf{y}:\mathsf{A}} \mathsf{a} = \mathsf{y}$, which means A is contractible. Hence, the only information this space encodes is the existence or nonexistence of a point up to continuous deformation (i.e. the space behaves like a truth value).

Convention 1.8.5 (Implicit casting of mere propositions)**.** Instead of $(\mathsf{A}, \mathsf{p}) : \mathbf{Prop}_i$, we will write $\mathsf{A} : \mathbf{Prop}_i$. When $\mathsf{A} : \mathbf{Prop}_i$ is an assumption in some lemma, we will interpret this as: "We have $\mathsf{A} : \mathcal{U}_i$, *and* an implicit proof that it is a mere proposition". When we want to treat $\mathsf{A} : \mathbf{Prop}_i$ as a pair, we will explicitly write it as such.

▲

Any inhabited proposition is contractible, as expressed by the following lemma.

Lemma 1.8.6. *Let* $\mathsf{A} : \mathbf{Prop}_i$ *be a proposition. If* A *is inhabited, then* A *is contractible.*

Proof. A is inhabited by hypothesis, so there is a point $\mathsf{w} : \mathsf{A}$.

Since A is a proposition, type $\prod_{\mathsf{y}:\mathsf{A}} \mathsf{w} = \mathsf{y}$ is inhabited. Hence, w acts as a center of contraction.

□

For the next lemma we require a definition.

Definition 1.8.7 (Logical equivalence)**.** Let $\mathsf{A} : \mathcal{U}_i$ and $\mathsf{B} : \mathcal{U}_j$ be two types. If there are functions:

$$\mathsf{f} : \mathsf{A} \to \mathsf{B}$$
$$\mathsf{g} : \mathsf{B} \to \mathsf{A}$$

then we say that A and B are *logically equivalent*. Hence, under the Curry-Howard correspondence, this reads: "A if and only if B".

▲

In general, logical equivalence of A and B does not imply that A and B are equivalent types. However, when A and B are mere propositions, this is indeed the case.

Lemma 1.8.8. *Let* $\mathsf{A} : \mathbf{Prop}_i$ *and* $\mathsf{B} : \mathbf{Prop}_j$ *be two propositions. If* A *and* B *are logically equivalent, then they are equivalent types.*

Proof. By hypothesis, A and B are logically equivalent, so, we are given two functions:

$$\mathsf{f} : \mathsf{A} \to \mathsf{B}$$
$$\mathsf{g} : \mathsf{B} \to \mathsf{A}$$

It remains to show that these functions are inverses of each other.

Let $\mathsf{w} : \mathsf{A}$. The equation $\mathsf{g}\,(\mathsf{f}\,\mathsf{w}) = \mathsf{w}$ holds because $\mathsf{g}\,(\mathsf{f}\,\mathsf{w})$ and w are terms of type A, which is a proposition (hence, both terms are identifiable).

Similarly, given $\mathsf{w} : \mathsf{B}$, the equation $\mathsf{f}\,(\mathsf{g}\,\mathsf{w}) = \mathsf{w}$ holds because $\mathsf{f}\,(\mathsf{g}\,\mathsf{w})$ and w are terms of type B, which is a proposition.

□

Now, we present the concept of set.

Definition 1.8.9 (Set)**.** Let $\mathsf{A} : \mathcal{U}_i$ be a type. We define type:

$$\mathsf{IsSet}\ \mathsf{A} :\equiv \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} \prod_{\mathsf{p},\mathsf{q}:\mathsf{w}=\mathsf{y}} \mathsf{p} = \mathsf{q}$$

If type $\mathsf{IsSet}\ \mathsf{A}$ is inhabited, we say that A is a *set*. Sets are also called 0-*types*. Notice that the definition of $\mathsf{IsSet}\ \mathsf{A}$ can also be given as:

$$\mathsf{IsSet}\ \mathsf{A} :\equiv \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} \mathsf{IsProp}\ (\mathsf{w} = \mathsf{y})$$

We define type:

$$\mathbf{Set}_i :\equiv \sum_{\mathsf{A}:\mathcal{U}_i} \mathsf{IsSet}\ \mathsf{A}$$

called the *universe of sets* at level i.

▲

A set is a type where any two of its terms can be identified in at most one way. This conforms with standard mathematical practice where it is proof-irrelevant whether or not two elements in a set are equal.

Under the homotopy interpretation, a set is a space where any two paths with the same end points can be *continuously* assigned a homotopy. Sets can also be interpreted as spaces where each path-connected component is contractible [20]. Therefore, if we contract each path-connected component, the result will be a space that looks like a discrete space, hence the name *set*.

Convention 1.8.10 (Implicit casting of sets)**.** Instead of $(\mathsf{A}, \mathsf{p}) : \mathbf{Set}_i$, we will write $\mathsf{A} : \mathbf{Set}_i$. When $\mathsf{A} : \mathbf{Set}_i$ is an assumption in some lemma, we will interpret this as: "We have $\mathsf{A} : \mathcal{U}_i$, *and* an implicit proof that it is a set". When we want to treat $\mathsf{A} : \mathbf{Set}_i$ as a pair, we will explicitly write it as such.

▲

Now, we present some results. The first one states that any mere proposition is a set.

Lemma 1.8.11. *If* $\mathsf{A} : \mathbf{Prop}_i$*, then* $\mathsf{A} : \mathbf{Set}_i$.

Proof. Since A is a proposition, there is a function $\mathsf{f} : \prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} \mathsf{w} = \mathsf{y}$.

First, we prove that there is a function:

$$\mathsf{g} : \prod_{\mathsf{w},\mathsf{y},\mathsf{z}:\mathsf{A}} \prod_{\mathsf{p}:\mathsf{y}=\mathsf{z}} \mathsf{p} = (\mathsf{f}\ \mathsf{w}\ \mathsf{y})^{-1} \cdot (\mathsf{f}\ \mathsf{w}\ \mathsf{z})$$

Let $\mathsf{w} : \mathsf{A}$. Then, by the induction principle for identity types, it is enough to prove:

$$\prod_{\mathsf{a}:\mathsf{A}} \mathsf{refl}_{\mathsf{A}}\ \mathsf{a} = (\mathsf{f}\ \mathsf{w}\ \mathsf{a})^{-1} \cdot (\mathsf{f}\ \mathsf{w}\ \mathsf{a})$$

So, let $\mathsf{a} : \mathsf{A}$. But type $\mathsf{refl}_{\mathsf{A}}\ \mathsf{a} = (\mathsf{f}\ \mathsf{w}\ \mathsf{a})^{-1} \cdot (\mathsf{f}\ \mathsf{w}\ \mathsf{a})$ is inhabited by Lemma 1.6.5(iii).

Therefore, function g exists. Now, to prove that A is a set, let $\mathsf{w}, \mathsf{y} : \mathsf{A}$ with $\mathsf{p}, \mathsf{q} : \mathsf{w} = \mathsf{y}$.

Then, we have:

$$\mathsf{g}\ \mathsf{w}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p} : \mathsf{p} = (\mathsf{f}\ \mathsf{w}\ \mathsf{w})^{-1} \cdot (\mathsf{f}\ \mathsf{w}\ \mathsf{y})$$
$$\mathsf{g}\ \mathsf{w}\ \mathsf{w}\ \mathsf{y}\ \mathsf{q} : \mathsf{q} = (\mathsf{f}\ \mathsf{w}\ \mathsf{w})^{-1} \cdot (\mathsf{f}\ \mathsf{w}\ \mathsf{y})$$

which means $(\mathsf{g}\ \mathsf{w}\ \mathsf{w}\ \mathsf{y}\ \mathsf{p}) \cdot (\mathsf{g}\ \mathsf{w}\ \mathsf{w}\ \mathsf{y}\ \mathsf{q})^{-1} : \mathsf{p} = \mathsf{q}$.

□

Lemma 1.8.11 has the following consequence.

Lemma 1.8.12. *Let* $\mathsf{A} : \mathcal{U}_i$*. Then,* $\mathsf{IsProp}\ \mathsf{A}$ *is a mere proposition.*

Proof. Let $\mathsf{f}, \mathsf{g} : \mathsf{IsProp}\ \mathsf{A}$. By function extensionality applied twice, to prove $\mathsf{f} = \mathsf{g}$ it is enough to prove $\mathsf{f}\ \mathsf{w}\ \mathsf{y} = \mathsf{g}\ \mathsf{w}\ \mathsf{y}$ for any $\mathsf{w}, \mathsf{y} : \mathsf{A}$.

So, let $\mathsf{w}, \mathsf{y} : \mathsf{A}$. Since A is a proposition, by Lemma 1.8.11 it is also a set. This means that type $\mathsf{w} = \mathsf{y}$ is a mere proposition (see alternative definition of set).

But we have $\mathsf{f}\ \mathsf{w}\ \mathsf{y} : \mathsf{w} = \mathsf{y}$ and $\mathsf{g}\ \mathsf{w}\ \mathsf{y} : \mathsf{w} = \mathsf{y}$, which means that $\mathsf{f}\ \mathsf{w}\ \mathsf{y} = \mathsf{g}\ \mathsf{w}\ \mathsf{y}$ is inhabited because $\mathsf{w} = \mathsf{y}$ is a mere proposition.

□

Another consequence of Lemma 1.8.11 is the following.

Lemma 1.8.13. *Let* $\mathsf{A} : \mathbf{Prop}_i$ *be a proposition, and* $\mathsf{w}, \mathsf{y} : \mathsf{A}$*. Then, type* $\mathsf{w} = \mathsf{y}$ *is contractible.*

Proof. Since A is a proposition, there is a function $\mathsf{f} : \prod_{\mathsf{r},\mathsf{t}:\mathsf{A}} \mathsf{r} = \mathsf{t}$.
Also, by Lemma 1.8.11, type A is a set. This means that $\mathsf{w} = \mathsf{y}$ is a mere proposition.
Since type $\mathsf{w} = \mathsf{y}$ is inhabited by $(\mathsf{f}\ \mathsf{w}\ \mathsf{y})$, it follows by Lemma 1.8.6 that type $\mathsf{w} = \mathsf{y}$ is contractible.

□

The next property provides a characterization for identities on Σ-types when the type family $\mathsf{P} : \mathsf{A} \to \mathcal{U}_j$ is a family of propositions $\mathsf{P} : \mathsf{A} \to \mathbf{Prop}_j$, also called a "predicate on A".

Lemma 1.8.14. *Let* $\mathsf{A} : \mathcal{U}_i$ *be a type and* $\mathsf{P} : \mathsf{A} \to \mathbf{Prop}_j$ *a family of propositions. Then, the following type is inhabited:*

$$\prod_{\mathsf{y},\mathsf{z}:\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} (\mathsf{y} = \mathsf{z}) \simeq (\mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z})$$

Proof. Let $\mathsf{y}, \mathsf{z} : \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$. By Lemma 1.6.25, we have:

$$(\mathsf{y} = \mathsf{z}) \simeq \sum_{\mathsf{p}:\mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}} \mathsf{p}_*\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}$$

Notice that given arbitrary $\mathsf{p} : \mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}$, type $\mathsf{p}_*\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}$ is an identity type on the mere proposition $\mathsf{P}\ (\mathsf{pr}_1\ \mathsf{z})$. Therefore, by Lemma 1.8.13, type $\mathsf{p}_*\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z}$ is contractible for every p.
Therefore, by Lemma 1.8.3, we have:

$$\left(\sum_{\mathsf{p}:\mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z}} \mathsf{p}_*\ (\mathsf{pr}_2\ \mathsf{y}) = \mathsf{pr}_2\ \mathsf{z} \right) \simeq (\mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z})$$

Hence:

$$(\mathsf{y} = \mathsf{z}) \simeq (\mathsf{pr}_1\ \mathsf{y} = \mathsf{pr}_1\ \mathsf{z})$$

since equivalences are transitive by Lemma 1.6.21.

□

We also have a characterization for identities in the universe of propositions $\mathbf{Prop}_i$.

Lemma 1.8.15. *The following type is inhabited:*

$$\prod_{(\mathsf{P},\mathsf{p}):\mathbf{Prop}_i} \prod_{(\mathsf{Q},\mathsf{q}):\mathbf{Prop}_j} ((\mathsf{P},\mathsf{q}) =_{\mathbf{Prop}_{\max\{i,j\}}} (\mathsf{Q},\mathsf{q})) \simeq (\mathsf{P} \simeq \mathsf{Q})$$

Proof. Let $(\mathsf{P},\mathsf{p}) : \mathbf{Prop}_i$ and $(\mathsf{Q},\mathsf{q}) : \mathbf{Prop}_j$. By the cumulative universe rule, we have $\mathsf{P}, \mathsf{Q} : \mathcal{U}_{\max\{i,j\}}$. But $\mathsf{p} : \mathsf{IsProp}\ \mathsf{P}$ and $\mathsf{q} : \mathsf{IsProp}\ \mathsf{Q}$. Therefore, $(\mathsf{P},\mathsf{q}), (\mathsf{Q},\mathsf{q}) : \mathbf{Prop}_{\max\{i,j\}}$.
Now, by Lemma 1.8.12, for every type T, type $\mathsf{IsProp}\ \mathsf{T}$ is a mere proposition. Therefore, by definition, type $\mathbf{Prop}_{\max\{i,j\}}$ is a Σ-type over a family of propositions. So, by Lemma 1.8.14 we have:

$$((\mathsf{P},\mathsf{p}) =_{\mathbf{Prop}_{\max\{i,j\}}} (\mathsf{Q},\mathsf{q})) \simeq (\mathsf{P} =_{\mathcal{U}_{\max\{i,j\}}} \mathsf{Q})$$

But, by Lemma 1.6.32, we have:

$$(\mathsf{P} =_{\mathcal{U}_{\max\{i,j\}}} \mathsf{Q}) \simeq (\mathsf{P} \simeq \mathsf{Q})$$

Hence:

$$((\mathsf{P},\mathsf{p}) =_{\mathbf{Prop}_{\max\{i,j\}}} (\mathsf{Q},\mathsf{q})) \simeq (\mathsf{P} \simeq \mathsf{Q})$$

since equivalences are transitive by Lemma 1.6.21.

□

As we would expect, equivalences preserve propositions and sets.

Lemma 1.8.16. *Let* $\mathsf{A}:\mathcal{U}_i$ *and* $\mathsf{B}:\mathcal{U}_j$ *be types. Suppose that* A *and* B *are equivalent types. Then the following statements hold.*

(i) If A *is a proposition, then* B *is a proposition.*

(ii) If A *is a set, then* B *is a set.*

Proof. Since A and B are equivalent, by univalence we have a proof $\mathsf{q}:\mathsf{A}=\mathsf{B}$. If A is a proposition, we have a proof $\mathsf{G}:\mathsf{IsProp}\ \mathsf{A}$. Hence, $\mathsf{transport}^{(\lambda \mathsf{Y}.\ \mathsf{IsProp}\ \mathsf{Y})}\ \mathsf{q}\ \mathsf{G}:\mathsf{IsProp}\ \mathsf{B}$ is a proof that B is a proposition.

The proof is similar when A is a set.

□

Propositions have the following closure properties.

Lemma 1.8.17 (Closure properties for propositions)**.** *Given the condition on the left side of the following table, we will have the conclusion on the right side of the table.*

	Condition	Conclusion
(1)	$\mathsf{A}:\mathbf{Prop}_i$, $\mathsf{P}:\mathsf{A}\to\mathbf{Prop}_j$	$\left(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right):\mathbf{Prop}_{\max\{i,j\}}$
(2)	$\mathsf{A}:\mathcal{U}_i$, $\mathsf{P}:\mathsf{A}\to\mathbf{Prop}_j$	$\left(\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right):\mathbf{Prop}_{\max\{i,j\}}$
(3)		$\mathbb{0}:\mathbf{Prop}_i$
(4)		$\mathbb{1}:\mathbf{Prop}_i$
(5)	$\mathsf{A}:\mathcal{U}_i$	$\neg\mathsf{A}:\mathbf{Prop}_i$
(6)	$\mathsf{A}:\mathbf{Set}_i$, $\mathsf{w}:\mathsf{A}$, $\mathsf{y}:\mathsf{A}$	$(\mathsf{w}=\mathsf{y}):\mathbf{Prop}_i$
(7)	$\mathsf{A}:\mathbf{Prop}_i$, $\mathsf{w}:\mathsf{A}$, $\mathsf{y}:\mathsf{A}$	$(\mathsf{w}=\mathsf{y}):\mathbf{Prop}_i$

Proof. We prove each case as follows.

(1) Let $\mathsf{y},\mathsf{z}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}$. By Lemma 1.8.14, to prove $\mathsf{y}=\mathsf{z}$ it is enough to prove $\mathsf{pr}_1\ \mathsf{y}=\mathsf{pr}_1\ \mathsf{z}$ (since P is a family of propositions).

But $(\mathsf{pr}_1\ \mathsf{y})=_{\mathsf{A}}(\mathsf{pr}_1\ \mathsf{z})$ is inhabited, since A is a mere proposition.

Type $\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}$ is at level $\max\{i,j\}$ by Remark 1.5.29.

(2) Let $\mathsf{f},\mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}$. By function extensionality, to prove $\mathsf{f}=\mathsf{g}$ it is enough to prove $\mathsf{f}\ \mathsf{w}=\mathsf{g}\ \mathsf{w}$ for every $\mathsf{w}:\mathsf{A}$. So, let $\mathsf{w}:\mathsf{A}$. But $\mathsf{f}\ \mathsf{w}=_{(\mathsf{P}\ \mathsf{w})}\mathsf{g}\ \mathsf{w}$ is inhabited, since $\mathsf{P}\ \mathsf{w}$ is a mere proposition.

Type $\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}$ is at level $\max\{i,j\}$ by Remark 1.5.16.

(3) Type $\prod_{\mathsf{w},\mathsf{y}:\mathbb{0}}\mathsf{w}=\mathsf{y}$ is inhabited by a direct application of the induction principle for $\mathbb{0}$.

(4) Let $\mathsf{y},\mathsf{z}:\mathbb{1}$. By Lemma 1.6.2, we have a function $\mathsf{f}:\prod_{\mathsf{w}:\mathbb{1}}\mathsf{w}=\star$. Hence, $(\mathsf{f}\ \mathsf{y})\cdot(\mathsf{f}\ \mathsf{z})^{-1}:\mathsf{y}=\mathsf{z}$.

(5) Type $\neg\mathsf{A}$ denotes $\mathsf{A}\to\mathbb{0}$. By (3), $\mathbb{0}$ is a proposition. Therefore, type $\mathsf{A}\to\mathbb{0}$ is a proposition by (2).

Type $\neg\mathsf{A}$ is at level i because A and $\mathbb{0}$ are at level i.[47]

(6) By definition of set.

(7) By Lemma 1.8.11, A is also a set. Therefore, this case reduces to (6).

□

Notice that Lemma 1.8.17 left out the case for sum types $\mathsf{A}+\mathsf{B}$. A sum type represents a disjoint union. Hence, it will not be a path-connected space in general (i.e. not a mere proposition). For example, $\mathbb{1}+\mathbb{1}$ is a discrete space with two disconnected points, even though $\mathbb{1}$ is a mere proposition.

Sets have similar closure properties.

Lemma 1.8.18 (Closure properties for sets)**.** *Given the condition on the left side of the following table, we will have the conclusion on the right side of the table.*

[47] Type $\mathbb{0}$ is at every universe level.

	Condition	Conclusion
(1)	$\mathsf{A}:\mathbf{Set}_i$, $\mathsf{P}:\mathsf{A}\to\mathbf{Set}_j$	$\left(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right):\mathbf{Set}_{\max\{i,j\}}$
(2)	$\mathsf{A}:\mathcal{U}_i$, $\mathsf{P}:\mathsf{A}\to\mathbf{Set}_j$	$\left(\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}\right):\mathbf{Set}_{\max\{i,j\}}$
(3)	$\mathsf{A}:\mathbf{Set}_i$, $\mathsf{B}:\mathbf{Set}_j$	$\mathsf{A}+\mathsf{B}:\mathbf{Set}_{\max\{i,j\}}$
(4)		$\mathbb{0}:\mathbf{Set}_i$
(5)		$\mathbb{1}:\mathbf{Set}_i$
(6)		$\mathbb{N}:\mathbf{Set}_i$
(7)	$\mathsf{A}:\mathcal{U}_i$	$\neg\mathsf{A}:\mathbf{Set}_i$
(8)	$\mathsf{A}:\mathbf{Set}_i$, $\mathsf{w}:\mathsf{A}$, $\mathsf{y}:\mathsf{A}$	$(\mathsf{w}=\mathsf{y}):\mathbf{Set}_i$
(9)		$\mathbf{Prop}_i:\mathbf{Set}_{i+1}$

Proof. We prove each case as follows.

(1) Let $\mathsf{y},\mathsf{z}:\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}$. We will prove that type $\mathsf{y}=\mathsf{z}$ is a mere proposition. From this, it will follow that $\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}$ is a set (see Definition 1.8.9).

By Lemma 1.6.25, we have:

$$(\mathsf{y}=\mathsf{z})\simeq\sum_{\mathsf{p}:\mathsf{pr}_1\ \mathsf{y}=\mathsf{pr}_1\ \mathsf{z}}\mathsf{p}_*\ (\mathsf{pr}_2\ \mathsf{y})=\mathsf{pr}_2\ \mathsf{z}$$

Since we know that A is a set, it follows that type $(\mathsf{pr}_1\ \mathsf{y})=_{\mathsf{A}}(\mathsf{pr}_1\ \mathsf{z})$ is a mere proposition. Also, we know that for every $\mathsf{w}:\mathsf{A}$, type $\mathsf{P}\ \mathsf{w}$ is a set. Hence, type $\mathsf{p}_*\ (\mathsf{pr}_2\ \mathsf{y})=_{(\mathsf{P}\ (\mathsf{pr}_1\ \mathsf{z}))}\mathsf{pr}_2\ \mathsf{z}$ is a mere proposition for every p.

Therefore, by Lemma 1.8.17, type $\sum_{\mathsf{p}:\mathsf{pr}_1\ \mathsf{y}=\mathsf{pr}_1\ \mathsf{z}}\mathsf{p}_*\ (\mathsf{pr}_2\ \mathsf{y})=\mathsf{pr}_2\ \mathsf{z}$ is a mere proposition.

But propositions are preserved under equivalences (Lemma 1.8.16), which means that type $\mathsf{y}=\mathsf{z}$ is a mere proposition.

Type $\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}$ is at level $\max\{i,j\}$ by Remark 1.5.29.

(2) Let $\mathsf{f},\mathsf{g}:\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}$. We will prove that type $\mathsf{f}=\mathsf{g}$ is a mere proposition. From this, it will follow that $\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}$ is a set.

By Lemma 1.6.28, we have:

$$(\mathsf{f}=\mathsf{g})\simeq\prod_{\mathsf{w}:\mathsf{A}}\mathsf{f}\ \mathsf{w}=\mathsf{g}\ \mathsf{w}$$

We know that for every $\mathsf{w}:\mathsf{A}$, type $\mathsf{P}\ \mathsf{w}$ is a set. Hence, type $\mathsf{f}\ \mathsf{w}=_{(\mathsf{P}\ \mathsf{w})}\mathsf{g}\ \mathsf{w}$ is a mere proposition for every $\mathsf{w}:\mathsf{A}$.

Therefore, by Lemma 1.8.17, type $\prod_{\mathsf{w}:\mathsf{A}}\mathsf{f}\ \mathsf{w}=\mathsf{g}\ \mathsf{w}$ is a mere proposition.

But propositions are preserved under equivalences (Lemma 1.8.16), which means that type $\mathsf{f}=\mathsf{g}$ is a mere proposition.

Type $\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\ \mathsf{w}$ is at level $\max\{i,j\}$ by Remark 1.5.16.

(3) For this case, we make use of the following three functions whose proof we omit. The reader can find a proof of these functions on Section 2.12 in the HoTT book [20].

$$\mathsf{L}:\prod_{\mathsf{a}_1,\mathsf{a}_2:\mathsf{A}}((\mathsf{inl}\ \mathsf{a}_1=\mathsf{inl}\ \mathsf{a}_2)\simeq(\mathsf{a}_1=\mathsf{a}_2))$$

$$\mathsf{R}:\prod_{\mathsf{b}_1,\mathsf{b}_2:\mathsf{B}}((\mathsf{inr}\ \mathsf{b}_1=\mathsf{inr}\ \mathsf{b}_2)\simeq(\mathsf{b}_1=\mathsf{b}_2))$$

$$\mathsf{M}:\prod_{\mathsf{a}:\mathsf{A}}\prod_{\mathsf{b}:\mathsf{B}}((\mathsf{inl}\ \mathsf{a}=\mathsf{inr}\ \mathsf{b})\simeq\mathbb{0})$$

We need to prove that type $\prod_{\mathsf{r},\mathsf{t}:\mathsf{A}+\mathsf{B}}\mathsf{IsProp}\ (\mathsf{r}=\mathsf{t})$ is inhabited.

By the induction principle for sum types applied twice (Lemma 1.5.35), it is enough to prove the following four cases:

- $\mathsf{IsProp}\ (\mathsf{inl}\ \mathsf{l}_1=\mathsf{inl}\ \mathsf{l}_2)$ for $\mathsf{l}_1,\mathsf{l}_2:\mathsf{A}$. We know A is a set, which means $\mathsf{l}_1=\mathsf{l}_2$ is a mere proposition. Since equivalences preserve propositions, by function L above we get that $\mathsf{inl}\ \mathsf{l}_1=\mathsf{inl}\ \mathsf{l}_2$ is a mere proposition.

- $\mathsf{IsProp}\ (\mathsf{inl}\ \mathsf{l}_1 = \mathsf{inr}\ \mathsf{r}_2)$ for $\mathsf{l}_1 : \mathsf{A}$ and $\mathsf{r}_2 : \mathsf{B}$. By Lemma 1.8.17, $\mathbb{0}$ is a proposition. Since equivalences preserve propositions, by function M above we get that $\mathsf{inl}\ \mathsf{l}_1 = \mathsf{inr}\ \mathsf{r}_2$ is a proposition.
- $\mathsf{IsProp}\ (\mathsf{inr}\ \mathsf{r}_1 = \mathsf{inl}\ \mathsf{l}_2)$ for $\mathsf{r}_1 : \mathsf{B}$ and $\mathsf{l}_2 : \mathsf{A}$. By Lemma 1.8.17, $\mathbb{0}$ is a proposition. Since equivalences preserve propositions, by function M above we get that $\mathsf{inr}\ \mathsf{r}_1 = \mathsf{inl}\ \mathsf{l}_2$ is a proposition.[48]
- $\mathsf{IsProp}\ (\mathsf{inr}\ \mathsf{r}_1 = \mathsf{inr}\ \mathsf{r}_2)$ for $\mathsf{r}_1, \mathsf{r}_2 : \mathsf{B}$. We know B is a set, which means $\mathsf{r}_1 = \mathsf{r}_2$ is a mere proposition. Since equivalences preserve propositions, by function R above we get that $\mathsf{inr}\ \mathsf{r}_1 = \mathsf{inr}\ \mathsf{r}_2$ is a mere proposition.

Type $\mathsf{A} + \mathsf{B}$ is at level $\max\{i, j\}$ by Remark 1.5.37.

(4), (5), (7), and (8). Each one of these types is a proposition by Lemma 1.8.17. Therefore, they are also sets by Lemma 1.8.11.

(6) We omit the proof as it is not critical for the results on this report. The reader can find a proof at Example 3.1.4 in the HoTT book [20]. An alternative proof can be found at Theorem 7.2.6 in the HoTT book.

(9) Since $\mathbf{Prop}_i$ is a Σ-type, by Remark 1.5.29, its universe level will be the maximum between $\mathcal{U}_i$ and $\mathsf{IsProp}\ \mathsf{P}$ for every $\mathsf{P} : \mathcal{U}_i$. By the universe introduction rule on page 8, $\mathcal{U}_i$ is at level $i + 1$.

Now, given $\mathsf{P} : \mathcal{U}_i$, type $\mathsf{IsProp}\ \mathsf{P}$ is a Π-type, therefore, by Remark 1.5.16, its level will be the maximum between the level of P and the level of the identity type $\mathsf{w} = \mathsf{y}$ for every $\mathsf{w}, \mathsf{y} : \mathsf{P}$. Since identity types preserve universe level (see formation rule on page 45), we conclude that $\mathsf{IsProp}\ \mathsf{P}$ is at level i.

Therefore, $\mathbf{Prop}_i$ is a type at level $\max\{i + 1, i\} = i + 1$.

It remains to prove that $\mathbf{Prop}_i$ is a set. Let $(\mathsf{P}, \mathsf{p}), (\mathsf{Q}, \mathsf{q}) : \mathbf{Prop}_i$. Again, we will prove that type $(\mathsf{P}, \mathsf{p}) =_{\mathbf{Prop}_i} (\mathsf{Q}, \mathsf{q})$ is a mere proposition.

By Lemma 1.8.15, we have:

$$((\mathsf{P}, \mathsf{p}) =_{\mathbf{Prop}_i} (\mathsf{Q}, \mathsf{q})) \simeq (\mathsf{P} \simeq \mathsf{Q})$$

But $\mathsf{P} \simeq \mathsf{Q}$ denotes type $\sum_{\mathsf{f}:\ \mathsf{P} \to \mathsf{Q}} \mathsf{IsEquiv}\ \mathsf{f}$, where $\mathsf{IsEquiv}\ \mathsf{f}$ denotes:

$$\left(\sum_{\mathsf{g}:\mathsf{Q}\to\mathsf{P}} \mathsf{f} \circ \mathsf{g} \sim \mathsf{id}_{\mathsf{Q}}\right) \times \left(\sum_{\mathsf{h}:\mathsf{Q}\to\mathsf{P}} \mathsf{h} \circ \mathsf{f} \sim \mathsf{id}_{\mathsf{P}}\right)$$

Since P and Q are propositions by hypothesis, all these types are mere propositions by closure properties in Lemma 1.8.17.

So, $(\mathsf{P}, \mathsf{p}) =_{\mathbf{Prop}_i} (\mathsf{Q}, \mathsf{q})$ is a proposition, as equivalences preserve them.

□

For the next lemma, we require a definition.

Definition 1.8.19 (Subtypes and subsets)**.** Let $\mathsf{A} : \mathcal{U}_i$ be a type and $\mathsf{P} : \mathsf{A} \to \mathbf{Prop}_j$ a family of propositions.

Type:

$$\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w} \tag{1.24}$$

will be called the *subtype* of A determined by P. If in addition A is a set, then type (1.24) is a set (Lemma 1.8.18) and it will be called the *subset* of A determined by P.

We sometimes abuse terminology and say that the family P *is* the subtype (subset, resp.) of A. This will cause no confusion, as P is not a type, but a only a term that determines type (1.24).

Therefore, we can safely say that $\mathsf{A} \to \mathbf{Prop}_j$ is the *type of all subtypes* of A or the *powertype* of A. Similarly, when A is a set, we can say that $\mathsf{A} \to \mathbf{Prop}_j$ is the *type of all subsets* of A or the *powerset* of A.

Notice that $\mathbf{Prop}_j$ is a set by Lemma 1.8.18, which means, by the same lemma, that $\mathsf{A} \to \mathbf{Prop}_j$ is *always* a set, independently of A, justifying the name "powerset".

▲

[48] Strictly speaking, if we use M, we only get that $\mathsf{inl}\ \mathsf{l}_2 = \mathsf{inr}\ \mathsf{r}_1$ is a proposition.
Although we did not state it as a lemma, one can prove that the symmetry operator (for any $\mathsf{C} : \mathcal{U}_k$):

$$^{-1} : \prod_{\mathsf{a},\mathsf{b}:\mathsf{C}} (\mathsf{a} = \mathsf{b}) \to (\mathsf{b} = \mathsf{a})$$

is an equivalence, by proving that it is its own inverse (with aid from Lemma 1.6.5(v)).
Hence $\mathsf{inl}\ \mathsf{l}_2 = \mathsf{inr}\ \mathsf{r}_1$ is equivalent to $\mathsf{inr}\ \mathsf{r}_1 = \mathsf{inl}\ \mathsf{l}_2$, and the result follows.

The next lemma states that no type is equivalent to its powertype. This is a generalization to Cantor's Theorem: there is no bijection between a set and its powerset.

Lemma 1.8.20. *Let* $\mathsf{A} : \mathcal{U}_i$ *be a type. Then the following type is inhabited:*

$$\neg((\mathsf{A} \to \mathbf{Prop}_j) \simeq \mathsf{A})$$

Proof. Suppose $(\mathsf{A} \to \mathbf{Prop}_j) \simeq \mathsf{A}$ is inhabited, we want to prove that $\mathbb{0}$ is inhabited.

By hypothesis, we have an equivalence $\mathsf{e} : (\mathsf{A} \to \mathbf{Prop}_j) \to \mathsf{A}$. Now, define:

$$\mathsf{k} :\equiv (\lambda \mathsf{a} : \mathsf{A}.\ \neg(\mathsf{e}^{-1}\ \mathsf{a}\ \mathsf{a})) : \mathsf{A} \to \mathbf{Prop}_j$$

Notice k is well-typed, since $\mathsf{e}^{-1}\ \mathsf{a}\ \mathsf{a} : \mathbf{Prop}_j$ and the negation operator produces a proposition at the same level (Lemma 1.8.17).

Since e is an equivalence, there is a proof $\mathsf{p}_1 : \mathsf{k} = \mathsf{e}^{-1}\ (\mathsf{e}\ \mathsf{k})$.

Define $\delta :\equiv \mathsf{e}\ \mathsf{k}$. Hence:[49]

$$\mathsf{p}_2 :\equiv \mathsf{happly}\ \mathsf{k}\ (\mathsf{e}^{-1}\ \delta)\ \mathsf{p}_1 : \mathsf{k} \sim \mathsf{e}^{-1}\ \delta$$

which means:

$$\mathsf{p}_2\ \delta : \mathsf{k}\ \delta = \mathsf{e}^{-1}\ \delta\ \delta$$

or equivalently:

$$\mathsf{p}_2\ \delta : \neg(\mathsf{e}^{-1}\ \delta\ \delta) = \mathsf{e}^{-1}\ \delta\ \delta$$

since $\mathsf{k}\ \delta \equiv \neg(\mathsf{e}^{-1}\ \delta\ \delta)$.

Now, we prove the following claim.

Claim 1: There is a function:

$$\mathsf{f} : \prod_{\mathsf{C}:\mathcal{U}_m} ((\neg\mathsf{C} = \mathsf{C}) \to \mathbb{0})$$

Suppose $\mathsf{C} : \mathcal{U}_m$ and $\mathsf{h}_1 : \neg\mathsf{C} = \mathsf{C}$. We want to prove that $\mathbb{0}$ is inhabited. We have:[50]

$$\mathsf{idtoequiv}\ \neg\mathsf{C}\ \mathsf{C}\ \mathsf{h}_1 : \neg\mathsf{C} \simeq \mathsf{C}$$

This means there are two functions $\mathsf{g}_1 : \neg\mathsf{C} \to \mathsf{C}$ and $\mathsf{g}_2 : \mathsf{C} \to \neg\mathsf{C}$. But then we can define:

$$\mathsf{g}_3 :\equiv (\lambda \mathsf{w} : \mathsf{C}.\ \mathsf{g}_2\ \mathsf{w}\ \mathsf{w}) : \neg\mathsf{C}$$

which means:

$$\mathsf{g}_3\ (\mathsf{g}_1\ \mathsf{g}_3) : \mathbb{0}$$

This proves the claim.

To finish the proof, by Claim 1 we have:

$$\mathsf{f}\ (\mathsf{e}^{-1}\ \delta\ \delta)\ (\mathsf{p}_2\ \delta) : \mathbb{0}$$

□

1.8.1 The propositional resizing axiom

Since universes are cumulative, we will have a function $\mathbf{Prop}_i \to \mathbf{Prop}_{i+1}$ defined as follows.

Definition 1.8.21 (Propositional resizing function)**.** We define a function:

$$\mathsf{PropR}_i : \mathbf{Prop}_i \to \mathbf{Prop}_{i+1}$$

by recursion on Σ-types as:

$$\mathsf{PropR}_i\ (\mathsf{P}, \mathsf{q}) :\equiv (\mathsf{P}, \mathsf{q})$$

Notice PropR_i is well-typed, since $\mathsf{P} : \mathcal{U}_i$ implies $\mathsf{P} : \mathcal{U}_{i+1}$ by the cumulative universe rule.

▲

[49] Function happly is defined in Lemma 1.6.27.

[50] Function idtoequiv is stated in Definition 1.6.31.

The propositional resizing axiom states that PropR_i is an equivalence.

- Propositional resizing (S-context: Γ):

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathsf{propres} : \mathsf{IsEquiv}\ \mathsf{PropR}_i}\ \text{PROP-RES-}i$$

  Here, $\mathsf{propres}$ is a term acting as proof to the fact that PropR_i is an equivalence. In informal proofs we will say "by propositional resizing..." when we want to use the inverse of PropR_i without explicit mention to the PROP-RES-i rule.

Propositional resizing implies that all propositional universes are mutually equivalent. This axiom will allow *impredicative* definitions. An impredicative definition is one that quantifies over a totality of objects to which the defined object belongs to. An example of an impredicative definition can be found in Example 2.2.8 of Chapter 2.

Also, the reader may have noticed that the powertype $\mathsf{A} \to \mathbf{Prop}_j$ is dependent on the universe level j. This means that there is a powertype for each universe level j. Since propositional resizing implies that all propositional universes are equivalent, it is irrelevant which level j we choose for $\mathsf{A} \to \mathbf{Prop}_j$. In other words, we are justified in calling $\mathsf{A} \to \mathbf{Prop}_j$ *the* powertype of A.

The propositional resizing axiom is not part of the standard presentation of HoTT, but it can be safely added to HoTT without producing an inconsistency [20].

We will use the propositional resizing axiom in a more general form as follows.

Lemma 1.8.22 (Generalized propositional resizing)**.** *If $i \leq j$, there is an equivalence:*

$$\mathsf{GPropR}_i^j : \mathbf{Prop}_i \to \mathbf{Prop}_j$$

such that for any $\mathsf{A} : \mathcal{U}_i$ and $\mathsf{p} : \mathsf{IsProp}\ \mathsf{A}$, it satisfies:

$$\mathsf{GPropR}_i^j\ (\mathsf{A}, \mathsf{p}) \equiv (\mathsf{A}, \mathsf{p})$$

Proof. If $i = j$, our equivalence is the identity function (Lemma 1.6.19). If $j = i + 1$, our equivalence is PropR_i. If $j > i + 1$, just compose the corresponding PropR function $j - i$ times, as equivalences compose by Lemma 1.6.21. □

If we resize a proposition using $(\mathsf{GPropR}_i^j)^{-1}$, we get an equivalent proposition.

Lemma 1.8.23. *Let $i \leq j$. The following type is inhabited:*

$$\prod_{(\mathsf{P},\mathsf{q}):\mathbf{Prop}_j} \mathsf{pr}_1\ ((\mathsf{GPropR}_i^j)^{-1}\ (\mathsf{P}, \mathsf{q})) \simeq \mathsf{P} \tag{1.25}$$

However, type (1.25) looks ugly due to the excessive use of pairs, parentheses, and the projection function. Hence, we can use Convention 1.8.5 to rewrite it as:

$$\prod_{\mathsf{P}:\mathbf{Prop}_j} ((\mathsf{GPropR}_i^j)^{-1}\ \mathsf{P}) \simeq \mathsf{P}$$

where the proof for $\mathsf{IsProp}\ \mathsf{P}$ is implicitly added wherever is needed.

Proof. Let $(\mathsf{P}, \mathsf{q}) : \mathbf{Prop}_j$. Define $\delta :\equiv ((\mathsf{GPropR}_i^j)^{-1}\ (\mathsf{P}, \mathsf{q})) : \mathbf{Prop}_i$.

Since GPropR_i^j is an equivalence, there is a proof $\mathsf{q}_1 : \mathsf{GPropR}_i^j\ \delta =_{\mathbf{Prop}_j} (\mathsf{P}, \mathsf{q})$.

But, by Lemma 1.6.1, we have a proof $\mathsf{q}_2 : (\mathsf{pr}_1\ \delta, \mathsf{pr}_2\ \delta) =_{\mathbf{Prop}_i} \delta$. Therefore:

$$\mathsf{q}_3 :\equiv \mathsf{transport}^{\mathsf{R}}\ \mathsf{q}_2^{-1}\ \mathsf{q}_1 : \mathsf{GPropR}_i^j\ (\mathsf{pr}_1\ \delta, \mathsf{pr}_2\ \delta) =_{\mathbf{Prop}_j} (\mathsf{P}, \mathsf{q})$$

where:

$$\mathsf{R} :\equiv \lambda \mathsf{A} : \mathbf{Prop}_i.\ \mathsf{GPropR}_i^j\ \mathsf{A} =_{\mathbf{Prop}_j} (\mathsf{P}, \mathsf{q})$$

However, by definition of GPropR_i^j, we have $\mathsf{GPropR}_i^j\ (\mathsf{pr}_1\ \delta, \mathsf{pr}_2\ \delta) \equiv (\mathsf{pr}_1\ \delta, \mathsf{pr}_2\ \delta) : \mathbf{Prop}_j$, which means:

$$\mathsf{q}_3 : (\mathsf{pr}_1\ \delta, \mathsf{pr}_2\ \delta) =_{\mathbf{Prop}_j} (\mathsf{P}, \mathsf{q})$$

Therefore, by Lemma 1.8.15, there is a proof:

$$\mathsf{q}_4 : \mathsf{pr}_1\ \delta \simeq \mathsf{P}$$

or by definition of δ:

$$\mathsf{q}_4 : \mathsf{pr}_1\ ((\mathsf{GPropR}_i^j)^{-1}\ (\mathsf{P}, \mathsf{q})) \simeq \mathsf{P}$$

□

We also have the following useful property.

Lemma 1.8.24. *Let $i \leq j$ be universe levels, $\mathsf{P} : \mathbf{Prop}_j$ a proposition, and $\mathsf{A} : \mathcal{U}_k$ a type.*
If $\mathsf{P} \to \mathsf{A}$ is inhabited, then $((\mathsf{GPropR}_i^j)^{-1}\ \mathsf{P}) \to \mathsf{A}$ is inhabited.

Proof. We are given a function $\mathsf{f} : \mathsf{P} \to \mathsf{A}$.
By Lemma 1.8.23, there is an equivalence:

$$\mathsf{e} : ((\mathsf{GPropR}_i^j)^{-1}\ \mathsf{P}) \to \mathsf{P}$$

Hence, $\mathsf{f} \circ \mathsf{e} : ((\mathsf{GPropR}_i^j)^{-1}\ \mathsf{P}) \to \mathsf{A}$.

□

1.9 Inductive types and initial algebras

The intuitive idea behind an inductive type is that *all* its elements are constructed from the "bottom up". An inductive type has base constructors for the bottom elements, and inductive constructors, which take previously constructed elements and return new elements. For example, the natural numbers have 0 as their base constructor and the successor function S as their inductive constructor.

In addition, inductive types have inductive and/or recursion principles, which allow the construction of functions *out* of the inductive type. These principles encode the idea that the constructors are the *only* way to build elements in the inductive type. For example, to define a function with domain the natural numbers, it is enough to define it on the zero case and the successor case, since these are the only ways to construct a natural number.

There are many ways to present an inductive type. One way is to provide formation, introduction, elimination, and computation rules as we did in Section 1.5.[51] More alternative ways can be found in Chapter 5 of the HoTT book [20], but the way we will be particularly interested is through the concept of *initial algebra*. Initial algebras will encode inductive types [20].

The most general definition of initial algebra comes from category theory. This report will use an special case of the general definition. In order the understand the special case, we need to have a look at how initial algebras are defined in category theory. For the moment, we will only state the definitions of these categorical concepts. We defer their explanation until we start modeling them in HoTT.

We start with the concept of category, as defined in Awodey [5].

Definition 1.9.1 (Category)**.** A *category* consists of:

- *Objects*: A, B, C, ...
- *Arrows*: f, g, h, ...

[51] The reason why most of the types in Section 1.5 have the name "ind" on their eliminators (see their elimination rules) is because these types are actually inductive types.
However, most of these types do not have inductive constructors, only base constructors (see their introduction rules). There is even a case of an inductive type (the empty type $\mathbb{0}$) without constructors at all.

- For each arrow f, there are given objects:

$$dom(f) \qquad cod(f)$$

  called the *domain* and *codomain* of f. Whenever an arrow is written as:

$$f : A \to B$$

  it is understood that A is $dom(f)$ and B is $cod(f)$.

- Given arrows $f : A \to B$ and $g : B \to C$, there is given an arrow:

$$g \circ f : A \to C$$

  called the *composite* of f and g.

- For each object A, there is given an arrow:

$$1_A : A \to A$$

  called the *identity arrow* of A.

such that the following properties are satisfied:

- Associativity:

$$h \circ (g \circ f) = (h \circ g) \circ f$$

  for all $f : A \to B$, $g : B \to C$, and $h : C \to D$.

- Unit:

$$f \circ 1_A = f = 1_B \circ f$$

  for all $f : A \to B$.

▲

Next comes the concept of functor, also defined in [5].

Definition 1.9.2 (Functor)**.** Let $\mathcal{C}$ and $\mathcal{D}$ be two categories. A *functor* H between categories $\mathcal{C}$ and $\mathcal{D}$, denoted $H : \mathcal{C} \to \mathcal{D}$, is a mapping of objects and arrows from $\mathcal{C}$ to $\mathcal{D}$, i.e.

- For each object A in $\mathcal{C}$, functor H maps A to an object in $\mathcal{D}$, denoted $H(A)$.
- For each arrow $f : A \to B$ in $\mathcal{C}$, functor H maps f to an arrow in $\mathcal{D}$, denoted $H(f)$, but in such a way that the domain of $H(f)$ is $H(A)$, and the codomain of $H(f)$ is $H(B)$. This is denoted as $H(f) : H(A) \to H(B)$.

so that the following conditions are satisfied:

$$H(1_A) = 1_{H(A)} \tag{1.26}$$
$$H(g \circ f) = H(g) \circ H(f) \tag{1.27}$$

for any object A and arrows $f : B \to C$ and $g : C \to D$ in $\mathcal{C}$.

▲

We can now define what an algebra is (see [1]).

Definition 1.9.3 (Algebra)**.** Let $\mathcal{C}$ be a category and $H : \mathcal{C} \to \mathcal{C}$ a functor.

An *H-algebra* is an object A in $\mathcal{C}$ together with an arrow $In_A : H(A) \to A$. We will denote H-algebras as (A, In_A).

▲

To be able to define what an initial algebra is, we need the notion of algebra morphism (see [1]).

Definition 1.9.4 (Algebra morphism)**.** Let $\mathcal{C}$ be a category and $H : \mathcal{C} \to \mathcal{C}$ a functor.

An *algebra morphism* between H-algebras (A, In_A) and (B, In_B) is an arrow $m : A \to B$ such that the following diagram commutes:

$$\begin{array}{ccc} H(A) & \xrightarrow{H(m)} & H(B) \\ {\scriptstyle In_A}\downarrow & & \downarrow{\scriptstyle In_B} \\ A & \xrightarrow[m]{} & B \end{array}$$

▲

And now we can state the definition of initial algebra (see [1]).

Definition 1.9.5 (Initial algebra)**.** Let $\mathcal{C}$ be a category and $H : \mathcal{C} \to \mathcal{C}$ a functor.

An *initial H-algebra* is an H-algebra (A, In_A) such that for any other H-algebra (B, In_B), there is a *unique* algebra morphism from (A, In_A) to (B, In_B).

▲

On this report, we will be interested on constructing initial/final (co)algebras inside *type universes*, which means that we will not work in arbitrary categories. However, observe that a type universe $\mathcal{U}_i$ can be understood as a category, where the objects are the types in the universe, the arrows are all functions between those types, arrow composition is just the function composition operator of Definition 1.5.13, the identity arrow is just the identity function of Definition 1.5.6, and the associativity and unit laws are satisfied by Lemmas 1.6.3 and 1.6.4.

Hence, the reader can consider all the following definitions to be just the general definitions above but instantiated on the "type universe categories".

The first notion we need to model in HoTT is that of functor.

Definition 1.9.6 (Functor)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_j$ be a function between type universes. Any term of type:

$$\mathsf{FunctorStr\ H} :\equiv \sum_{\mathsf{map}:\prod_{\mathsf{A,B}:\mathcal{U}_i} (\mathsf{A}\to\mathsf{B})\to(\mathsf{H\ A})\to(\mathsf{H\ B})} \left[\left(\prod_{\mathsf{A}:\mathcal{U}_i} \mathsf{map\ A\ A\ id_A} \sim \mathsf{id_{(H\ A)}}\right) \times \left(\prod_{\mathsf{A,B,C}:\mathcal{U}_i} \prod_{\mathsf{g}:\mathsf{A}\to\mathsf{B}} \prod_{\mathsf{h}:\mathsf{B}\to\mathsf{C}} \mathsf{map\ A\ C\ (h \circ g)} \sim \mathsf{(map\ B\ C\ h) \circ (map\ A\ B\ g)}\right)\right]$$

will be called a *functor structure* for H. Also, any term of type:

$$\sum_{\mathsf{G}:\mathcal{U}_i\to\mathcal{U}_j} \mathsf{FunctorStr\ G}$$

will be called a *functor*.

▲

A functor $(\ \mathsf{H}, (\ \mathsf{map},\ (\alpha, \beta)\ )\ ) : \sum_{\mathsf{G}:\mathcal{U}_i\to\mathcal{U}_j} \mathsf{FunctorStr\ G}$ is a tuple consisting on a function H between type universes (i.e. the function maps objects between "type universe categories"), together with a map function (i.e. each arrow $f : A \to B$ is mapped to an arrow $map(f) : H(A) \to H(B)$) in such a way that α is a proof that the arrow mapping respects the identity function (i.e. Condition (1.26) in Definition 1.9.2) and β is a proof that the arrow mapping respects function composition (i.e. Condition (1.27) in Definition 1.9.2).

Notice that a functor was defined using pointwise identifiability ($\sim$) instead of the identity type ($=$), but it does not matter which one is used, because we have function extensionality.

Now, a convention regarding functors.

Convention 1.9.7. In informal arguments, we will say: "Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_j$ be a functor..." leaving implicit all the functor structure. When we want to refer to the map function associated with H, it will be written as $\mathsf{map}^{\mathsf{H}}$.

Also, given $\mathsf{A}, \mathsf{B} : \mathcal{U}_i$ and $\mathsf{f} : \mathsf{A} \to \mathsf{B}$, instead of $\mathsf{map}^{\mathsf{H}}\ \mathsf{A}\ \mathsf{B}\ \mathsf{f}$, we will write $\mathsf{map}^{\mathsf{H}}\ \mathsf{f}$, where parameters A and B are left implicit.

These conventions also apply when defining functors.

▲

For example, we can define a functor by first defining the type mapping function:

$$\mathsf{NatF}_i :\equiv (\lambda \mathsf{Y}{:}\mathcal{U}_i.\ \mathbb{1} + \mathsf{Y}) : \mathcal{U}_i \to \mathcal{U}_i$$

And then define its arrow mapping function:

$$\mathsf{map}^{\mathsf{NatF}_i} : \prod_{\mathsf{A},\mathsf{B}:\mathcal{U}_i} (\mathsf{A} \to \mathsf{B}) \to ((\mathbb{1} + \mathsf{A}) \to (\mathbb{1} + \mathsf{B}))$$

or equivalently:

$$\mathsf{map}^{\mathsf{NatF}_i} : \prod_{\mathsf{A},\mathsf{B}:\mathcal{U}_i} (\mathsf{A} \to \mathsf{B}) \to ((\mathsf{NatF}_i\ \mathsf{A}) \to (\mathsf{NatF}_i\ \mathsf{B}))$$

by the recursion principle for sum types (for arbitrary $\mathsf{A}, \mathsf{B} : \mathcal{U}_i$, $\mathsf{f} : \mathsf{A} \to \mathsf{B}$, $\mathsf{l} : \mathbb{1}$, and $\mathsf{r} : \mathsf{A}$):

$$\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{f}\ (\mathsf{inl}\ \mathsf{l}) :\equiv \mathsf{inl}\ \mathsf{l}$$
$$\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{f}\ (\mathsf{inr}\ \mathsf{r}) :\equiv \mathsf{inr}\ (\mathsf{f}\ \mathsf{r})$$

See Lemma 3.2.29 for a proof that NatF_i (together with its arrow mapping function) defines a functor.

We will be interested on two properties a functor may have.

Definition 1.9.8 (Endofunctor)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_j$ be a functor. We will say that H is an *endofunctor*, if $j = i$.

In informal arguments we will say: "Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor..." as it will be implied that it is an endofunctor, because we are using the same universe level on the domain and codomain of H.

▲

Definition 1.9.9 (Set-preserving functor)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_j$ be a functor. We will say that H *preserves sets* or that H is a *set-preserving functor* if given a set $\mathsf{C} : \mathbf{Set}_i$, we have that $\mathsf{H}\ \mathsf{C}$ is also a set.

This property will play a role until Chapter 3, but we define it in here for completeness.

▲

Functors preserve equivalences, as the following lemma shows.

Lemma 1.9.10. *Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_j$ be a functor, and $k, m \leq i$ universe levels. Let $\mathsf{A} : \mathcal{U}_k$ and $\mathsf{B} : \mathcal{U}_m$ be two types, and $\mathsf{e} : \mathsf{A} \to \mathsf{B}$ an equivalence.*

Then, $\mathsf{map}^{\mathsf{H}}\ \mathsf{e} : (\mathsf{H}\ \mathsf{A}) \to (\mathsf{H}\ \mathsf{B})$ is an equivalence.[52]

Proof. By Lemma 1.6.16 we need to construct an inverse for $\mathsf{map}^{\mathsf{H}}\ \mathsf{e}$. We claim that $\mathsf{map}^{\mathsf{H}}\ \mathsf{e}^{-1}$ will be its inverse.

Type $\prod_{\mathsf{a}:\mathsf{H}\ \mathsf{A}} \mathsf{map}^{\mathsf{H}}\ \mathsf{e}^{-1}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{e}\ \mathsf{a}) = \mathsf{a}$ is inhabited, since we have:[53]

$$\begin{aligned}
\mathsf{map}^{\mathsf{H}}\ \mathsf{e}^{-1}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{e}\ \mathsf{a}) &\overset{(1)}{=} \mathsf{map}^{\mathsf{H}}\ (\mathsf{e}^{-1} \circ \mathsf{e})\ \mathsf{a} \\
&\overset{(2)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{a} \\
&\overset{(3)}{=} \mathsf{id}_{(\mathsf{H}\ \mathsf{A})}\ \mathsf{a} \\
&\overset{(4)}{=} \mathsf{a}
\end{aligned}$$

where (1) follows by functorial mapping of function composition (see Definition 1.9.6), (2) follows by function extensionality applied to the fact that e is an equivalence,[54] (3) by functorial mapping on the identity function, and (4) by definition of the identity function.

Type $\prod_{\mathsf{b}:\mathsf{H}\ \mathsf{B}} \mathsf{map}^{\mathsf{H}}\ \mathsf{e}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{e}^{-1}\ \mathsf{b}) = \mathsf{b}$ is also inhabited by an identical argument.

□

[52] Notice that levels k, m need to be lower or equal to i because H needs to be applied on types A and B.

[53] Notice that these identities can be pasted together using the transitivity operator defined in Lemma 1.5.60, i.e. this sequence of identities are really claiming that type $\mathsf{map}^{\mathsf{H}}\ \mathsf{e}^{-1}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{e}\ \mathsf{a}) = \mathsf{a}$ is inhabited.

[54] In more detail, we know there is a proof $\mathsf{q}_1 : (\mathsf{e}^{-1} \circ \mathsf{e}) \sim \mathsf{id}_{\mathsf{A}}$ since e is an equivalence. Therefore, by function extensionality, there is a proof $\mathsf{q}_2 : \mathsf{e}^{-1} \circ \mathsf{e} = \mathsf{id}_{\mathsf{A}}$. Hence, by the application operator, we have $\mathsf{ap}_{\mathsf{map}^{\mathsf{H}}}\ \mathsf{q}_2 : \mathsf{map}^{\mathsf{H}}\ (\mathsf{e}^{-1} \circ \mathsf{e}) = \mathsf{map}^{\mathsf{H}}\ \mathsf{id}_{\mathsf{A}}$.

Now, apply the happly function defined in Lemma 1.6.27 to get a proof for $\mathsf{map}^{\mathsf{H}}\ (\mathsf{e}^{-1} \circ \mathsf{e})\ \mathsf{a} = \mathsf{map}^{\mathsf{H}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{a}$.

Now, we introduce the concept of algebra.

Definition 1.9.11 (H-algebra)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, and $k \leq i$ be a universe level.[55] Any term of type:

$$\mathsf{Alg}_k\ \mathsf{H} :\equiv \sum_{\mathsf{A}:\mathcal{U}_k} (\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$$

will be called an H-*algebra* at level k. In other words, an algebra is a type $\mathsf{A} : \mathcal{U}_k$ together with a function $(\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$. Functions of type $(\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$ are usually called *In functions*.

Also, notice that functor H is an endofunctor. This conforms with Definition 1.9.3, where the functor is required to be a mapping between the *same* category.

▲

For example, a NatF_i-algebra looks like:

$$\sum_{\mathsf{A}:\mathcal{U}_k} ((\mathbb{1} + \mathsf{A}) \to \mathsf{A})$$

which can be rewritten as:[56]

$$\sum_{\mathsf{A}:\mathcal{U}_k} (\mathsf{A} \times (\mathsf{A} \to \mathsf{A}))$$

In other words, a NatF_i-algebra is a type with a distinguished element and a unary operator. The natural numbers are an example of a NatF_i-algebra, because their "In" function $\mathsf{In}_{\mathbb{N}} : (\mathsf{NatF}_i\ \mathbb{N}) \to \mathbb{N}$ can be defined by the recursion principle for sum types:

$$\begin{aligned} \mathsf{In}_{\mathbb{N}}\ (\mathsf{inl}\ \mathsf{l}) &:\equiv 0 \\ \mathsf{In}_{\mathbb{N}}\ (\mathsf{inr}\ \mathsf{n}) &:\equiv \mathsf{succ}\ \mathsf{n} \end{aligned} \tag{1.28}$$

Given an initial algebra A, the intuition behind its "In" function $\mathsf{In}_{\mathsf{A}} : (\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$ is that it will encode the *constructors* for the inductive type. For example, Equations (1.28) show that the constructors for $\mathbb{N}$ (0 and succ) can be "wrapped inside" the $\mathsf{In}_{\mathbb{N}} : (\mathsf{NatF}_i\ \mathbb{N}) \to \mathbb{N}$ function.

Next, we define the concept of algebra morphism.

Definition 1.9.12 (Algebra morphism)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, and $k, l \leq i$ universe levels. Let $(\mathsf{A}, \mathsf{In}_{\mathsf{A}}) : \mathsf{Alg}_k\ \mathsf{H}$, and $(\mathsf{B}, \mathsf{In}_{\mathsf{B}}) : \mathsf{Alg}_l\ \mathsf{H}$ be two H-algebras. Any term of type:

$$\mathsf{AlgMor}\ \mathsf{A}\ \mathsf{B} :\equiv \sum_{\mathsf{f}:\mathsf{A}\to\mathsf{B}} (\mathsf{f} \circ \mathsf{In}_{\mathsf{A}}) \sim (\mathsf{In}_{\mathsf{B}} \circ \mathsf{map}^{\mathsf{H}}\ \mathsf{f})$$

will be called an H-*algebra morphism* between H-algebras $(\mathsf{A}, \mathsf{In}_{\mathsf{A}})$ and $(\mathsf{B}, \mathsf{In}_{\mathsf{B}})$.

In other words, function $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ will be an H-algebra morphism between $(\mathsf{A}, \mathsf{In}_{\mathsf{A}})$ and $(\mathsf{B}, \mathsf{In}_{\mathsf{B}})$, if the following diagram commutes:

$$\begin{array}{ccc} \mathsf{H}\ \mathsf{A} & \xrightarrow{\mathsf{map}^{\mathsf{H}}\ \mathsf{f}} & \mathsf{H}\ \mathsf{B} \\ {\scriptstyle \mathsf{In}_{\mathsf{A}}}\downarrow & & \downarrow{\scriptstyle \mathsf{In}_{\mathsf{B}}} \\ \mathsf{A} & \xrightarrow[\mathsf{f}]{} & \mathsf{B} \end{array}$$

▲

For example, if $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ is a NatF_i-algebra morphism, it satisfies:

$$\prod_{\mathsf{w}:\mathbb{1}+\mathsf{A}} \mathsf{f}\ (\mathsf{In}_{\mathsf{A}}\ \mathsf{w}) = \mathsf{In}_{\mathsf{B}}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{f}\ \mathsf{w})$$

[55] Notice that level k needs to be lower or equal to i because H needs to be applied on type A.

[56] It can be proved that for any type B, types $(\mathbb{1} + \mathsf{B}) \to \mathsf{B}$ and $\mathsf{B} \times (\mathsf{B} \to \mathsf{B})$ are equivalent, but we omit the proof.

which reduces to the following equations, when w is given in canonical form:

$$\begin{aligned} \mathsf{f}\ (\mathsf{In_A}\ (\mathsf{inl}\ \star)) &= \mathsf{In_B}\ (\mathsf{inl}\ \star) \\ \prod_{\mathsf{a}:\mathsf{A}} \mathsf{f}\ (\mathsf{In_A}\ (\mathsf{inr}\ \mathsf{a})) &= \mathsf{In_B}\ (\mathsf{inr}\ (\mathsf{f}\ \mathsf{a})) \end{aligned}$$

If we define $\mathsf{D_A} :\equiv \mathsf{In_A}\ (\mathsf{inl}\ \star)$ to be the distinguished element of A (and similarly for B), and $\mathsf{U_A}\ \mathsf{a} :\equiv \mathsf{In_A}\ (\mathsf{inr}\ \mathsf{a})$ to be the unary operator on A (and similarly for B), then, these equations can be rewritten as:

$$\begin{aligned} \mathsf{f}\ \mathsf{D_A} &= \mathsf{D_B} \\ \prod_{\mathsf{a}:\mathsf{A}} \mathsf{f}\ (\mathsf{U_A}\ \mathsf{a}) &= \mathsf{U_B}\ (\mathsf{f}\ \mathsf{a}) \end{aligned}$$

which state that f preserves the distinguished element and the unary operator. In fact, when A is $\mathbb{N}$, these equations look like (use function $\mathsf{In}_{\mathbb{N}}$ as defined by Equations (1.28)):

$$\begin{aligned} \mathsf{f}\ 0 &= \mathsf{D_B} \\ \prod_{\mathsf{a}:\mathbb{N}} \mathsf{f}\ (\mathsf{succ}\ \mathsf{a}) &= \mathsf{U_B}\ (\mathsf{f}\ \mathsf{a}) \end{aligned}$$

which correspond to the definition of $\mathsf{f}\colon \mathbb{N} \to \mathsf{B}$ by *recursion*.

Therefore, algebra morphisms will encode *recursive equations* or *computation rules*.

Also, H-algebra morphisms compose as one would expect.

Lemma 1.9.13. *Let $\mathsf{H}\colon \mathcal{U}_i \to \mathcal{U}_i$ be a functor. Let $k, l, m \leq i$ be universe levels, $(\mathsf{A}, \mathsf{In_A})\colon \mathsf{Alg}_k\ \mathsf{H}$, $(\mathsf{B}, \mathsf{In_B})\colon \mathsf{Alg}_l\ \mathsf{H}$, and $(\mathsf{C}, \mathsf{In_C})\colon \mathsf{Alg}_m\ \mathsf{H}$ three H-algebras.*

Let $\mathsf{f}\colon \mathsf{A} \to \mathsf{B}$, and $\mathsf{g}\colon \mathsf{B} \to \mathsf{C}$ be two H-algebra morphisms. Then, $\mathsf{g} \circ \mathsf{f}\colon \mathsf{A} \to \mathsf{C}$ is an H-algebra morphism.

In other words, if both squares:

$$\begin{array}{ccccc} \mathsf{H\ A} & \xrightarrow{\mathsf{map^H\ f}} & \mathsf{H\ B} & \xrightarrow{\mathsf{map^H\ g}} & \mathsf{H\ C} \\ \downarrow{\scriptstyle \mathsf{In_A}} & & \downarrow{\scriptstyle \mathsf{In_B}} & & \downarrow{\scriptstyle \mathsf{In_C}} \\ \mathsf{A} & \xrightarrow[\mathsf{f}]{} & \mathsf{B} & \xrightarrow[\mathsf{g}]{} & \mathsf{C} \end{array}$$

commute, then the following square commutes:

$$\begin{array}{ccc} \mathsf{H\ A} & \xrightarrow{\mathsf{map^H\ (g \circ f)}} & \mathsf{H\ C} \\ \downarrow{\scriptstyle \mathsf{In_A}} & & \downarrow{\scriptstyle \mathsf{In_C}} \\ \mathsf{A} & \xrightarrow[\mathsf{g \circ f}]{} & \mathsf{C} \end{array}$$

Proof. Let $\mathsf{w}\colon \mathsf{H\ A}$. Then:

$$\begin{aligned} \mathsf{In_C}\ (\mathsf{map^H}\ (\mathsf{g} \circ \mathsf{f})\ \mathsf{w}) &\overset{(1)}{=} \mathsf{In_C}\ (\mathsf{map^H}\ \mathsf{g}\ (\mathsf{map^H}\ \mathsf{f}\ \mathsf{w})) \\ &\overset{(2)}{=} \mathsf{g}\ (\mathsf{In_B}\ (\mathsf{map^H}\ \mathsf{f}\ \mathsf{w})) \\ &\overset{(3)}{=} \mathsf{g}\ (\mathsf{f}\ (\mathsf{In_A}\ \mathsf{w})) \\ &\overset{(4)}{=} (\mathsf{g} \circ \mathsf{f})\ (\mathsf{In_A}\ \mathsf{w}) \end{aligned}$$

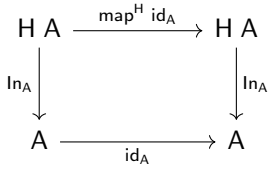
where (1) follows by functorial mapping of H,[57] (2) holds because g is a morphism, (3) holds because f is a morphism, and (4) follows by definition of function composition.

□

Also, the identity function is an H-algebra morphism.

Lemma 1.9.14. *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor. Let* $k \leq i$ *be a universe level, and* $(\mathsf{A}, \mathsf{In}_\mathsf{A}) : \mathsf{Alg}_k\ \mathsf{H}$ *an* H*-algebra. Then,* $\mathsf{id}_\mathsf{A} : \mathsf{A} \to \mathsf{A}$ *is an* H*-algebra morphism.*

In other words, the following square commutes:



Proof. Let $\mathsf{w} : \mathsf{H}\ \mathsf{A}$. Then:

$$\begin{aligned}\mathsf{In}_\mathsf{A}\ (\mathsf{map}^\mathsf{H}\ \mathsf{id}_\mathsf{A}\ \mathsf{w}) &\overset{(1)}{=} \mathsf{In}_\mathsf{A}\ (\mathsf{id}_{(\mathsf{H}\ \mathsf{A})}\ \mathsf{w}) \\ &\overset{(2)}{=} \mathsf{In}_\mathsf{A}\ \mathsf{w} \\ &\overset{(3)}{=} \mathsf{id}_\mathsf{A}\ (\mathsf{In}_\mathsf{A}\ \mathsf{w})\end{aligned}$$

where (1) follows by functorial mapping on the identity function, and (2) and (3) by definition of the identity function.

□

Next, we introduce the concept of initial algebra.

Definition 1.9.15 (Initial H-algebra)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor. Let $k \leq i$ be a universe level. Any term of type:

$$\mathsf{InAlg}_k\ \mathsf{H} :\equiv \sum_{(\mathsf{A},\mathsf{In}_\mathsf{A}):\mathsf{Alg}_k\ \mathsf{H}}\ \prod_{(\mathsf{B},\mathsf{In}_\mathsf{B}):\mathsf{Alg}_k\ \mathsf{H}} \mathsf{IsContr}\ (\mathsf{AlgMor}\ \mathsf{A}\ \mathsf{B})$$

will be called an *initial* H*-algebra* at level k. If type $\mathsf{InAlg}_k\ \mathsf{H}$ is inhabited, we say that functor H *has an initial algebra*.

▲

In other words, an initial algebra is an algebra $(\mathsf{A}, \mathsf{In}_\mathsf{A})$ such that for any other algebra $(\mathsf{B}, \mathsf{In}_\mathsf{B})$ there is a *unique* function $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ satisfying the recursive equations, i.e. a *unique* algebra morphism from A to B (in HoTT terminology, the type of morphisms $\mathsf{AlgMor}\ \mathsf{A}\ \mathsf{B}$ is contractible, see Definition 1.8.1).

An initial algebra encapsulates the following ideas:

- The constructors for A encoded in function In_A.
- A recursion principle *and* its computation rules, both encoded in type $\prod_{(\mathsf{B},\mathsf{In}_\mathsf{B}):\mathsf{Alg}_k\ \mathsf{H}} \mathsf{IsContr}\ (\mathsf{AlgMor}\ \mathsf{A}\ \mathsf{B})$. The conditions for the recursion principle are encoded in the fact that $(\mathsf{B}, \mathsf{In}_\mathsf{B})$ is an algebra.

An initial algebra *also* encodes an *induction principle*, though it is less obvious (see page 65 in [17] and Section 5.5 in the HoTT book [20] for examples).

Also, any two initial algebras for a functor H can be proved to be equivalent (see Lemma 2.5.5 in [17]). So, we are justified in calling it *the* initial algebra for H.

[57] In more detail, by functorial mapping of function composition (see Definition 1.9.6), we have a proof:

$$\mathsf{q} : \mathsf{map}^\mathsf{H}\ (\mathsf{g} \circ \mathsf{f})\ \mathsf{w} = \mathsf{map}^\mathsf{H}\ \mathsf{g}\ (\mathsf{map}^\mathsf{H}\ \mathsf{f}\ \mathsf{w})$$

which implies:

$$\mathsf{ap}_{\mathsf{In}_\mathsf{C}}\ \mathsf{q} : \mathsf{In}_\mathsf{C}\ (\mathsf{map}^\mathsf{H}\ (\mathsf{g} \circ \mathsf{f})\ \mathsf{w}) = \mathsf{In}_\mathsf{C}\ (\mathsf{map}^\mathsf{H}\ \mathsf{g}\ (\mathsf{map}^\mathsf{H}\ \mathsf{f}\ \mathsf{w}))$$

by the application operator ap (see Lemma 1.5.62).

Notice that algebra morphisms encode computation rules through the use of identity types (=). To the contrary, all computation rules in Section 1.5 use definitional equality (≡). This is why initial algebras are called *homotopy initial algebras* in HoTT [20].

As a final point, beware that type $\mathsf{InAlg}_k\ \mathsf{H}$ may not be inhabited for an arbitrary functor H. We will explore the initial algebra existence question in Chapter 3.

1.10 Coinductive types and final coalgebras

The intuitive idea behind a coinductive type is that all its elements are "decomposable" into an "observation" and a "next state", where this next state may be decomposable again. For example, we can imagine an infinite list (also called a *stream*) to be a structure decomposable into a first element (i.e. the observation) and the rest of the list (i.e. the next state). The rest of the list is again an infinite list that is decomposable again into an observation and a next state, and so on ad infinitum.

Inductive types have constructors, while coinductive types have *destructors*, which are in charge of decomposing the elements into observations and next states. For the streams example, one destructor is the *head* operation returning the first element of the list, and the second destructor is the *tail* operation returning the list without the first element.

In addition, coinductive types encode corecursion principles, which allow the construction of functions *into* the coinductive type. These principles encode the idea that the destructors are the *only* way to decompose elements on the coinductive type. For example, to define a function into the type of streams, it is enough to state what is the head and tail of the image of every input to the function, because using the head and tail destructors is the only way to decompose a stream in the first place.

Inductive types are encoded by initial algebras, while coinductive types are encoded by *final coalgebras* [4].

As we did for initial algebras, the categorical definitions are presented first, followed by their definitions in HoTT. We start with the concept of coalgebra (see [1]).

Definition 1.10.1 (Coalgebra)**.** Let $\mathcal{C}$ be a category and $H : \mathcal{C} \to \mathcal{C}$ a functor.

An *H-coalgebra* is an object A in $\mathcal{C}$ together with an arrow $Out_A : A \to H(A)$. We will denote H-coalgebras as (A, Out_A)

▲

We also have the notion of coalgebra morphism [1].

Definition 1.10.2 (Coalgebra morphism)**.** Let $\mathcal{C}$ be a category and $H : \mathcal{C} \to \mathcal{C}$ a functor.

A *coalgebra morphism* between H-coalgebras (A, Out_A) and (B, Out_B) is an arrow $m : A \to B$ such that the following diagram commutes:

$$\begin{array}{ccc} A & \xrightarrow{m} & B \\ {\scriptstyle Out_A}\downarrow & & \downarrow{\scriptstyle Out_B} \\ H(A) & \xrightarrow[H(m)]{} & H(B) \end{array}$$

▲

And next, the concept of final coalgebra [1].

Definition 1.10.3 (Final coalgebra)**.** Let $\mathcal{C}$ be a category and $H : \mathcal{C} \to \mathcal{C}$ a functor.

A *final H-coalgebra* is an H-coalgebra (A, Out_A) such that for any other H-coalgebra (B, Out_B), there is a *unique* coalgebra morphism from (B, Out_B) to (A, Out_A).

▲

Now, it is time to model these concepts in HoTT. We start with the concept of coalgebra.

Definition 1.10.4 (H-coalgebra)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, and $k \leq i$ a universe level. Any term of type:

$$\mathsf{CoAlg}_k\ \mathsf{H} :\equiv \sum_{\mathsf{A}:\mathcal{U}_k} \mathsf{A} \to (\mathsf{H}\ \mathsf{A})$$

will be called an H-*coalgebra* at level k. In other words, a coalgebra is a type $\mathsf{A} : \mathcal{U}_k$ together with a function $\mathsf{A} \to (\mathsf{H}\ \mathsf{A})$. Functions of type $\mathsf{A} \to (\mathsf{H}\ \mathsf{A})$ are usually called *Out functions*.

▲

To see an example of a coalgebra, let us first define a functor. Given $\mathsf{A} : \mathcal{U}_i$, define:

$$\mathsf{StreamF}_i\ \mathsf{A} :\equiv (\lambda \mathsf{Y} : \mathcal{U}_i.\ \mathsf{A} \times \mathsf{Y}) : \mathcal{U}_i \to \mathcal{U}_i$$

Now define its arrow mapping function:

$$\prod_{\mathsf{C},\mathsf{D}:\mathcal{U}_i} (\mathsf{C} \to \mathsf{D}) \to ((\mathsf{A} \times \mathsf{C}) \to (\mathsf{A} \times \mathsf{D}))$$

or equivalently:

$$\prod_{\mathsf{C},\mathsf{D}:\mathcal{U}_i} (\mathsf{C} \to \mathsf{D}) \to ((\mathsf{StreamF}_i\ \mathsf{A}\ \mathsf{C}) \to (\mathsf{StreamF}_i\ \mathsf{A}\ \mathsf{D}))$$

by recursion on Σ-types (for any $\mathsf{C}, \mathsf{D} : \mathcal{U}_i$, $\mathsf{f} : \mathsf{C} \to \mathsf{D}$, $\mathsf{o} : \mathsf{A}$, and $\mathsf{s} : \mathsf{C}$):

$$\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{f}\ (\mathsf{o}, \mathsf{s}) :\equiv (\mathsf{o}, \mathsf{f}\ \mathsf{s})$$

Lemma 3.3.36 provides a proof that $\mathsf{StreamF}_i\ \mathsf{A}$ (together with its arrow mapping function) defines a functor.

Now, a $\mathsf{StreamF}_i\ \mathsf{A}$-coalgebra will look like:

$$\sum_{\mathsf{B}:\mathcal{U}_k} (\mathsf{B} \to (\mathsf{A} \times \mathsf{B}))$$

which can be rewritten as:[58]

$$\sum_{\mathsf{B}:\mathcal{U}_k} ((\mathsf{B} \to \mathsf{A}) \times (\mathsf{B} \to \mathsf{B}))$$

In other words, a $\mathsf{StreamF}_i\ \mathsf{A}$-coalgebra is a type with two operations. One operation returns elements on A (call it the observation function), while the other is a unary operator (call it the next state function).

To explain the nature of "Out" functions, let us suppose we were able to construct the type of streams $\mathbb{Str}_{\mathsf{A}}$ for arbitrary $\mathsf{A} : \mathcal{U}_i$ (this will be done in Section 3.3.4), together with the following two destructors for $\mathbb{Str}_{\mathsf{A}}$:

$$\mathsf{head} : \mathbb{Str}_{\mathsf{A}} \to \mathsf{A}$$
$$\mathsf{tail} : \mathbb{Str}_{\mathsf{A}} \to \mathbb{Str}_{\mathsf{A}}$$

Then, $\mathbb{Str}_{\mathsf{A}}$ will be a $\mathsf{StreamF}_i\ \mathsf{A}$-coalgebra if we define its "Out" function as follows:

$$\mathsf{Out}_{\mathbb{Str}_{\mathsf{A}}}\ \mathsf{s} :\equiv (\mathsf{head}\ \mathsf{s}, \mathsf{tail}\ \mathsf{s}) \tag{1.29}$$

Given a final coalgebra C, the intuition behind its "Out" function $\mathsf{Out}_{\mathsf{C}} : \mathsf{C} \to (\mathsf{H}\ \mathsf{C})$ is that it will encode the *destructors* for the coinductive type. For example, Equation (1.29) shows that the destructors for $\mathbb{Str}_{\mathsf{A}}$ (i.e. head and tail) can be "wrapped inside" the $\mathsf{Out}_{\mathbb{Str}_{\mathsf{A}}} : \mathbb{Str}_{\mathsf{A}} \to (\mathsf{StreamF}_i\ \mathsf{A}\ \mathbb{Str}_{\mathsf{A}})$ function.

Now, we define the concept of coalgebra morphism.

Definition 1.10.5 (Coalgebra morphism)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor. Let $k, l \leq i$ be universe levels, $(\mathsf{A}, \mathsf{Out}_{\mathsf{A}}) : \mathsf{CoAlg}_k\ \mathsf{H}$, and $(\mathsf{B}, \mathsf{Out}_{\mathsf{B}}) : \mathsf{CoAlg}_l\ \mathsf{H}$ two H-coalgebras. Any term of type:

$$\mathsf{CoAlgMor}\ \mathsf{A}\ \mathsf{B} :\equiv \sum_{\mathsf{f}:\mathsf{A}\to\mathsf{B}} (\mathsf{map}^{\mathsf{H}}\ \mathsf{f} \circ \mathsf{Out}_{\mathsf{A}}) \sim (\mathsf{Out}_{\mathsf{B}} \circ \mathsf{f})$$

will be called an H-*coalgebra morphism* between H-coalgebras $(\mathsf{A}, \mathsf{Out}_{\mathsf{A}})$ and $(\mathsf{B}, \mathsf{Out}_{\mathsf{B}})$.

[58] Given $\mathsf{C} : \mathcal{U}_i$ and $\mathsf{D} : \mathcal{U}_j$, it can be proved that $\mathsf{C} \to (\mathsf{D} \times \mathsf{C})$ and $(\mathsf{C} \to \mathsf{D}) \times (\mathsf{C} \to \mathsf{C})$ are equivalent types, but we omit the proof.

In other words, function $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ will be an H-coalgebra morphism between $(\mathsf{A}, \mathsf{Out}_\mathsf{A})$ and $(\mathsf{B}, \mathsf{Out}_\mathsf{B})$, if the following diagram commutes:

$$\begin{array}{ccc} \mathsf{A} & \xrightarrow{\mathsf{f}} & \mathsf{B} \\ {\scriptstyle \mathsf{Out}_\mathsf{A}}\downarrow & & \downarrow{\scriptstyle \mathsf{Out}_\mathsf{B}} \\ \mathsf{H}\,\mathsf{A} & \xrightarrow[\mathsf{map}^\mathsf{H}\,\mathsf{f}]{} & \mathsf{H}\,\mathsf{B} \end{array}$$

▲

For example, if $\mathsf{f} : \mathsf{C} \to \mathsf{D}$ is a $\mathsf{StreamF}_i\ \mathsf{A}$-coalgebra morphism, it satisfies:

$$\prod_{\mathsf{w}:\mathsf{C}} \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{f}\ (\mathsf{Out}_\mathsf{C}\ \mathsf{w}) = \mathsf{Out}_\mathsf{D}\ (\mathsf{f}\ \mathsf{w})$$

Since $\mathsf{Out}_\mathsf{C}\ \mathsf{w}$ has the type of a pair, by Lemma 1.6.1 we can rewrite this equation as:

$$\prod_{\mathsf{w}:\mathsf{C}} \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{f}\ (\mathsf{pr}_1\ (\mathsf{Out}_\mathsf{C}\ \mathsf{w}), \mathsf{pr}_2\ (\mathsf{Out}_\mathsf{C}\ \mathsf{w})) = \mathsf{Out}_\mathsf{D}\ (\mathsf{f}\ \mathsf{w})$$

which, by definition of $\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{f}$, simplifies to:

$$\prod_{\mathsf{w}:\mathsf{C}} (\mathsf{pr}_1\ (\mathsf{Out}_\mathsf{C}\ \mathsf{w}), \mathsf{f}\ (\mathsf{pr}_2\ (\mathsf{Out}_\mathsf{C}\ \mathsf{w}))) = \mathsf{Out}_\mathsf{D}\ (\mathsf{f}\ \mathsf{w})$$

Now, split $\mathsf{Out}_\mathsf{D}\ (\mathsf{f}\ \mathsf{w})$ into a pair, as we did for $\mathsf{Out}_\mathsf{C}\ \mathsf{w}$, to get:

$$\prod_{\mathsf{w}:\mathsf{C}} (\mathsf{pr}_1\ (\mathsf{Out}_\mathsf{C}\ \mathsf{w}), \mathsf{f}\ (\mathsf{pr}_2\ (\mathsf{Out}_\mathsf{C}\ \mathsf{w}))) = (\mathsf{pr}_1\ (\mathsf{Out}_\mathsf{D}\ (\mathsf{f}\ \mathsf{w})), \mathsf{pr}_2\ (\mathsf{Out}_\mathsf{D}\ (\mathsf{f}\ \mathsf{w})))$$

If we define $\mathsf{O}_\mathsf{C}\ \mathsf{w} :\equiv \mathsf{pr}_1\ (\mathsf{Out}_\mathsf{C}\ \mathsf{w})$ to be the observation function on C (and similarly for D), and $\mathsf{S}_\mathsf{C}\ \mathsf{w} :\equiv \mathsf{pr}_2\ (\mathsf{Out}_\mathsf{C}\ \mathsf{w})$ to be the next state function on C (and similarly for D), then, our equation is rewritten as:

$$\prod_{\mathsf{w}:\mathsf{C}} (\mathsf{O}_\mathsf{C}\ \mathsf{w}, \mathsf{f}\ (\mathsf{S}_\mathsf{C}\ \mathsf{w})) = (\mathsf{O}_\mathsf{D}\ (\mathsf{f}\ \mathsf{w}), \mathsf{S}_\mathsf{D}\ (\mathsf{f}\ \mathsf{w}))$$

This equation can be split into two equations using the ap operator with the projection functions pr_1 and pr_2 (see Lemma 1.5.62):

$$\prod_{\mathsf{w}:\mathsf{C}} \mathsf{O}_\mathsf{C}\ \mathsf{w} = \mathsf{O}_\mathsf{D}\ (\mathsf{f}\ \mathsf{w})$$
$$\prod_{\mathsf{w}:\mathsf{C}} \mathsf{f}\ (\mathsf{S}_\mathsf{C}\ \mathsf{w}) = \mathsf{S}_\mathsf{D}\ (\mathsf{f}\ \mathsf{w})$$

These equations express that f leaves intact the current observation in w, and that f preserves the next state of w. In fact, when D is the type of streams $\mathbb{Str}_\mathsf{A}$, these equations look like (use function $\mathsf{Out}_{\mathbb{Str}_\mathsf{A}}$ as defined by Equation (1.29)):

$$\prod_{\mathsf{w}:\mathsf{C}} \mathsf{O}_\mathsf{C}\ \mathsf{w} = \mathsf{head}\ (\mathsf{f}\ \mathsf{w})$$
$$\prod_{\mathsf{w}:\mathsf{C}} \mathsf{f}\ (\mathsf{S}_\mathsf{C}\ \mathsf{w}) = \mathsf{tail}\ (\mathsf{f}\ \mathsf{w})$$

which correspond to the definition of $\mathsf{f} : \mathsf{C} \to \mathbb{Str}_\mathsf{A}$ by *corecursion*, i.e. if we read the equations from right to left, they express how the image of each w under f can be decomposed as a stream.

Therefore, coalgebra morphisms will encode *corecursive equations* or *computation rules*.

H-coalgebra morphisms compose as one would expect.

Lemma 1.10.6. *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor. Let* $k, l, m \leq i$ *be universe levels,* $(\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{CoAlg}_k\ \mathsf{H}$, $(\mathsf{B}, \mathsf{Out}_\mathsf{B}) : \mathsf{CoAlg}_l\ \mathsf{H}$, *and* $(\mathsf{C}, \mathsf{Out}_\mathsf{C}) : \mathsf{CoAlg}_m\ \mathsf{H}$ *three* H*-coalgebras.*

Let $\mathsf{f} : \mathsf{A} \to \mathsf{B}$, *and* $\mathsf{g} : \mathsf{B} \to \mathsf{C}$ *be two* H*-coalgebra morphisms. Then,* $\mathsf{g} \circ \mathsf{f} : \mathsf{A} \to \mathsf{C}$ *is an* H*-coalgebra morphism.*

In other words, if both squares:

$$\begin{array}{ccccc} \mathsf{A} & \xrightarrow{\mathsf{f}} & \mathsf{B} & \xrightarrow{\mathsf{g}} & \mathsf{C} \\ \downarrow{\scriptstyle \mathsf{Out}_\mathsf{A}} & & \downarrow{\scriptstyle \mathsf{Out}_\mathsf{B}} & & \downarrow{\scriptstyle \mathsf{Out}_\mathsf{C}} \\ \mathsf{H}\ \mathsf{A} & \xrightarrow[\mathsf{map}^\mathsf{H}\ \mathsf{f}]{} & \mathsf{H}\ \mathsf{B} & \xrightarrow[\mathsf{map}^\mathsf{H}\ \mathsf{g}]{} & \mathsf{H}\ \mathsf{C} \end{array}$$

commute, then the following square commutes:

$$\begin{array}{ccc} \mathsf{A} & \xrightarrow{(\mathsf{g} \circ \mathsf{f})} & \mathsf{C} \\ \downarrow{\scriptstyle \mathsf{Out}_\mathsf{A}} & & \downarrow{\scriptstyle \mathsf{Out}_\mathsf{C}} \\ \mathsf{H}\ \mathsf{A} & \xrightarrow[\mathsf{map}^\mathsf{H}\ (\mathsf{g} \circ \mathsf{f})]{} & \mathsf{H}\ \mathsf{C} \end{array}$$

Proof. Let $\mathsf{w} : \mathsf{A}$. Then:

$$\begin{aligned} \mathsf{map}^\mathsf{H}\ (\mathsf{g} \circ \mathsf{f})\ (\mathsf{Out}_\mathsf{A}\ \mathsf{w}) &\overset{(1)}{=} \mathsf{map}^\mathsf{H}\ \mathsf{g}\ (\mathsf{map}^\mathsf{H}\ \mathsf{f}\ (\mathsf{Out}_\mathsf{A}\ \mathsf{w})) \\ &\overset{(2)}{=} \mathsf{map}^\mathsf{H}\ \mathsf{g}\ (\mathsf{Out}_\mathsf{B}\ (\mathsf{f}\ \mathsf{w})) \\ &\overset{(3)}{=} \mathsf{Out}_\mathsf{C}\ (\mathsf{g}\ (\mathsf{f}\ \mathsf{w})) \\ &\overset{(4)}{=} \mathsf{Out}_\mathsf{C}\ ((\mathsf{g} \circ \mathsf{f})\ \mathsf{w}) \end{aligned}$$

where (1) follows by functorial mapping of H (see Definition 1.9.6), (2) holds because f is a morphism, (3) holds because g is a morphism, and (4) follows by definition of function composition.

□

Also, the identity function is an H-coalgebra morphism.

Lemma 1.10.7. *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor. Let* $k \leq i$ *be a universe level, and* $(\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ *an* H*-coalgebra. Then,* $\mathsf{id}_\mathsf{A} : \mathsf{A} \to \mathsf{A}$ *is an* H*-coalgebra morphism.*

In other words, the following square commutes:

$$\begin{array}{ccc} \mathsf{A} & \xrightarrow{\mathsf{id}_\mathsf{A}} & \mathsf{A} \\ \downarrow{\scriptstyle \mathsf{Out}_\mathsf{A}} & & \downarrow{\scriptstyle \mathsf{Out}_\mathsf{A}} \\ \mathsf{H}\ \mathsf{A} & \xrightarrow[\mathsf{map}^\mathsf{H}\ \mathsf{id}_\mathsf{A}]{} & \mathsf{H}\ \mathsf{A} \end{array}$$

Proof. Let $\mathsf{w} : \mathsf{A}$. Then:

$$\begin{aligned} \mathsf{map}^\mathsf{H}\ \mathsf{id}_\mathsf{A}\ (\mathsf{Out}_\mathsf{A}\ \mathsf{w}) &\overset{(1)}{=} \mathsf{id}_{(\mathsf{H}\ \mathsf{A})}\ (\mathsf{Out}_\mathsf{A}\ \mathsf{w}) \\ &\overset{(2)}{=} \mathsf{Out}_\mathsf{A}\ \mathsf{w} \\ &\overset{(3)}{=} \mathsf{Out}_\mathsf{A}\ (\mathsf{id}_\mathsf{A}\ \mathsf{w}) \end{aligned}$$

where (1) follows by functorial mapping on the identity function (see Definition 1.9.6), (2) by definition of the identity function $\mathsf{id}_{(\mathsf{H}\ \mathsf{A})}$, and (3) by definition of the identity function id_A.

□

Next, we introduce the concept of final coalgebra.

Definition 1.10.8 (Final H-coalgebra)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor. Let $k \leq i$ be a universe level. Any term of type:

$$\mathsf{FinCoAlg}_k\ \mathsf{H} :\equiv \sum_{(\mathsf{A},\mathsf{Out}_\mathsf{A}):\mathsf{CoAlg}_k\ \mathsf{H}} \prod_{(\mathsf{B},\mathsf{Out}_\mathsf{B}):\mathsf{CoAlg}_k\ \mathsf{H}} \mathsf{IsContr}\ (\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{A})$$

will be called a *final* H-*coalgebra* at level k. If type $\mathsf{FinCoAlg}_k\ \mathsf{H}$ is inhabited, we say that functor H *has a final coalgebra*.

▲

In other words, a final coalgebra is a coalgebra $(\mathsf{A}, \mathsf{Out}_\mathsf{A})$ such that for any other coalgebra $(\mathsf{B}, \mathsf{Out}_\mathsf{B})$ there is a *unique* function $\mathsf{f} : \mathsf{B} \to \mathsf{A}$ satisfying the corecursive equations, i.e. a *unique* morphism from B to A (in HoTT terminology, the type of morphisms $\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{A}$ is contractible, see Definition 1.8.1).

A final coalgebra encapsulates the following ideas:

- The destructors for A encoded in function Out_A.
- Type $\prod_{(\mathsf{B},\mathsf{Out}_\mathsf{B}):\mathsf{CoAlg}_k\ \mathsf{H}} \mathsf{IsContr}\ (\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{A})$ encodes a corecursion principle *and* its computation rules. The conditions for the corecursion principle are encoded in the fact that $(\mathsf{B}, \mathsf{Out}_\mathsf{B})$ is a coalgebra.

A final coalgebra *also* encodes a *coinduction principle* (see Section 2.7 in [17] for an explanation). In addition, see Lemma 3.3.39 for an example of a coinduction principle.

Also, any two final coalgebras for a functor H can be proved to be equivalent (see Lemma 2.6.4 in [17]). So, we are justified in calling it *the* final coalgebra for H.

Notice that coalgebra morphisms encode computation rules through the use of identity types ($=$). Definitional equality ($\equiv$) is not used. This is why final coalgebras are called *homotopy final coalgebras* in HoTT.

As it happens with initial algebras, type $\mathsf{FinCoAlg}_k\ \mathsf{H}$ may not be inhabited for an arbitrary functor H. We will explore the final coalgebra existence question in Chapter 3.

1.11 Higher Inductive types

A higher inductive type (HIT) is an inductive type with the additional property of having *higher-order constructors* or just *higher constructors*.

Standard constructors introduce terms into the type, as we have seen in previous sections. Instead, higher constructors introduce *proofs for identities*.

From the point of view of the homotopy interpretation, standard constructors introduce individual points into the space, while higher constructors introduce *paths*, *surfaces*, or even higher dimensional constructions into the space.

It is possible to present HITs by using inference rules, as we have done so far for every type. However, the reader may have noticed that once an induction principle is proved, the elimination and computation rules are never used again. Also, the formation and introduction rules can always be stated in natural language. Therefore, it is not necessary to explicitly state inference rules when introducing a new type into HoTT, as it is enough to *state in natural language* how the type is formed, what are its constructors, and declare its induction principle without proof. This is the standard approach taken by the HoTT book. If the reader is curious about how HITs are presented with explicit inference rules, an example can be found in Appendix A.3.2 of the HoTT book [20].

Of course, whenever new types are introduced into the type theory, we need to augment the set of syntactical terms $\mathcal{TERM}$ (i.e. we need to add more clauses to Definition 1.2.1). Also, we need to define the appropriate bind structures for the new types. However, these are irrelevant technicalities, because we are only concerned with informal proofs in HoTT. Hence, we will omit these details from now on.[59]

This section presents the minimum amount of HITs required by the results on Chapters 2 and 3. But there are many more HITs that can be added into HoTT. See Chapter 6 in the HoTT book [20], where many more HITs are explained.

[59] The reader can fill the technical details. They are quite straightforward.

1.11.1 (-1)-truncations

Given a type $\mathsf{A}\colon \mathcal{U}_i$, the (-1)-truncation of A, denoted $\|\mathsf{A}\|$, is a type that identifies *all* terms in A. Because all terms in $\|\mathsf{A}\|$ are identified, type $\|\mathsf{A}\|$ is a mere proposition (see Definition 1.8.4).[60]

The name "truncation" comes from the homotopy interpretation. Because $\|\mathsf{A}\|$ is a mere proposition, space $\|\mathsf{A}\|$ loses all the non-trivial homotopy information that space A has, i.e. the homotopy information of A gets truncated.

Definition 1.11.1 (Formation and introduction rules for (-1)-truncations)**.** Let $\mathsf{A}\colon \mathcal{U}_i$ be a type. There is a type at level i, denoted $\|\mathsf{A}\|$, and called the (-1)-*truncation* of A.

Type $\|\mathsf{A}\|$ has the following two constructors:

- Term constructor. $|\ |\colon \mathsf{A} \to \|\mathsf{A}\|$.

  Instead of $|\ |\ \mathsf{a}$, we will write $|\mathsf{a}|$.

- Higher constructor. $|\mathsf{iden}|\colon \prod_{\mathsf{w},\mathsf{y}:\|\mathsf{A}\|} \mathsf{w} = \mathsf{y}$.

  Instead of $|\mathsf{iden}|\ \mathsf{w}\ \mathsf{y}$, we will write $|\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|$.

▲

Constructor $|\ |$ makes copies of all terms in A, while constructor $|\mathsf{iden}|$ introduces the proof $|\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|\colon \mathsf{w} = \mathsf{y}$ for any two terms w, y in $\|\mathsf{A}\|$. In other words, constructor $|\mathsf{iden}|$ is expressing that $\|\mathsf{A}\|$ is a mere proposition.

(-1)-truncations are subject to the following induction principle.

Definition 1.11.2 (Induction principle for (-1)-truncations)**.** Let $\mathsf{A}\colon \mathcal{U}_i$ be a type and $\mathsf{P}\colon \|\mathsf{A}\| \to \mathcal{U}_j$ a type family. If we can build two functions:

$$\mathsf{h}\colon \prod_{\mathsf{a}:\mathsf{A}} \mathsf{P}\ |\mathsf{a}|$$

$$\mathsf{k}\colon \prod_{\mathsf{w},\mathsf{y}:\|\mathsf{A}\|} \prod_{\mathsf{a}:\mathsf{P}\ \mathsf{w}} \prod_{\mathsf{b}:\mathsf{P}\ \mathsf{y}} \mathsf{transport}^{\mathsf{P}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|\ \mathsf{a} = \mathsf{b}$$

then we can define a function:

$$\mathsf{f}\colon \prod_{\mathsf{w}:\|\mathsf{A}\|} \mathsf{P}\ \mathsf{w}$$

by the following equations:[61]

$$\mathsf{f}\ |\mathsf{a}| :\equiv \mathsf{h}\ \mathsf{a}$$

$$\mathsf{apd}_{\mathsf{f}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}| = \mathsf{k}\ \mathsf{w}\ \mathsf{y}\ (\mathsf{f}\ \mathsf{w})\ (\mathsf{f}\ \mathsf{y})$$

where $\mathsf{a}\colon \mathsf{A}$ and $\mathsf{w}, \mathsf{y}\colon \|\mathsf{A}\|$.

▲

To understand the induction principle, it is useful to have a look at the general pattern for induction principles. So far, all induction principles define a function out of a type A by stating how the function behaves on canonical elements of A (canonical elements are those that can be put in the form of a constructor). For example:

- To define a function f with domain a Σ-type, by the induction principle (Lemma 1.5.24) we define f on pairs (a, b), as follows:

  $$\mathsf{f}\ (\mathsf{a}, \mathsf{b}) :\equiv \mathsf{g}\ \mathsf{a}\ \mathsf{b}$$

  where g is a given function.

  In other words:

  $$\mathsf{f}\ (\mathsf{a}, \mathsf{b}) :\equiv \text{ some function dependent on } \mathsf{a} \text{ and } \mathsf{b}$$

[60] The number (-1) suggests that the (-1)-truncation produces a mere proposition, because propositions are also called (-1)-types.

[61] Here, apd is the dependent application operator, see Lemma 1.5.63. Also, notice that the equation involving apd is well-typed, since $\mathsf{apd}_{\mathsf{f}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|\colon \mathsf{transport}^{\mathsf{P}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|\ (\mathsf{f}\ \mathsf{w}) = \mathsf{f}\ \mathsf{y}$, and $\mathsf{k}\ \mathsf{w}\ \mathsf{y}\ (\mathsf{f}\ \mathsf{w})\ (\mathsf{f}\ \mathsf{y})\colon \mathsf{transport}^{\mathsf{P}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|\ (\mathsf{f}\ \mathsf{w}) = \mathsf{f}\ \mathsf{y}$.

- To define a function f with domain a sum type, by the induction principle (Lemma 1.5.35) we define f on terms of the form $\mathsf{inl}\ \mathsf{a}$ and $\mathsf{inr}\ \mathsf{b}$, as follows:

$$\mathsf{f}\ (\mathsf{inl}\ \mathsf{a}) :\equiv \mathsf{g}\ \mathsf{a}$$
$$\mathsf{f}\ (\mathsf{inr}\ \mathsf{b}) :\equiv \mathsf{h}\ \mathsf{b}$$

  where g and h are given functions.

  In other words:

$$\mathsf{f}\ (\mathsf{inl}\ \mathsf{a}) :\equiv \text{ some function dependent on } \mathsf{a}$$
$$\mathsf{f}\ (\mathsf{inr}\ \mathsf{b}) :\equiv \text{ some function dependent on } \mathsf{b}$$

- To define a function f with domain the unit type, by the induction principle (Lemma 1.5.47) we define f on term $\star$, as follows:

$$\mathsf{f}\ \star :\equiv \mathsf{c}$$

  where c is a given term.

  In other words:

$$\mathsf{f}\ \star :\equiv \text{ some constant}$$

- To define a function f with domain the natural numbers, by the induction principle (Lemma 1.5.52) we define f on terms of the form 0 and $(\mathsf{succ}\ \mathsf{n})$, as follows:

$$\mathsf{f}\ 0 :\equiv \mathsf{c}$$
$$\mathsf{f}\ (\mathsf{succ}\ \mathsf{n}) :\equiv \mathsf{g}\ \mathsf{n}\ (\mathsf{f}\ \mathsf{n})$$

  where g is a given function and c a given term.

  In other words:

$$\mathsf{f}\ 0 :\equiv \text{ some constant}$$
$$\mathsf{f}\ (\mathsf{succ}\ \mathsf{n}) :\equiv \text{ some function dependent on } \mathsf{n} \text{ and recursive call } (\mathsf{f}\ \mathsf{n})$$

  Notice that the last equation also depends on the recursive call $(\mathsf{f}\ \mathsf{n})$. This seems to suggest that whenever an input variable has the type of the domain (like n in our example), a recursive call on the variable needs to be added to the function body (like $(\mathsf{f}\ \mathsf{n})$ in our example). Contrast this with all previous equations, where no input variable (a or b) has the type of the domain. Hence, there is no recursive call on those variables.

To further exemplify the pattern. Let us suppose that given types $\mathsf{B} : \mathcal{U}_i$ and $\mathsf{D} : \mathcal{U}_j$, we introduce type $\mathsf{Foo}_{\mathsf{B},\mathsf{D}} : \mathcal{U}_{\max\{i,j\}}$ into HoTT, with the following constructors:

$$\mathsf{c}_1 : \mathsf{Foo}_{\mathsf{B},\mathsf{D}}$$
$$\mathsf{c}_2 : \mathsf{B} \to \mathsf{Foo}_{\mathsf{B},\mathsf{D}}$$
$$\mathsf{c}_3 : \mathsf{Foo}_{\mathsf{B},\mathsf{D}} \to \mathsf{Foo}_{\mathsf{B},\mathsf{D}} \to \mathsf{D} \to \mathsf{Foo}_{\mathsf{B},\mathsf{D}}$$

Given terms $\mathsf{b} : \mathsf{B}$ and $\mathsf{d} : \mathsf{D}$, some examples of terms in type $\mathsf{Foo}_{\mathsf{B},\mathsf{D}}$ are:

$$\mathsf{c}_1 : \mathsf{Foo}_{\mathsf{B},\mathsf{D}}$$
$$\mathsf{c}_2\ \mathsf{b} : \mathsf{Foo}_{\mathsf{B},\mathsf{D}}$$
$$\mathsf{c}_3\ (\mathsf{c}_2\ \mathsf{b})\ \mathsf{c}_1\ \mathsf{d} : \mathsf{Foo}_{\mathsf{B},\mathsf{D}}$$

Now, to define a function f with domain $\mathsf{Foo}_{\mathsf{B},\mathsf{D}}$, we will have to define three clauses in the induction principle for $\mathsf{Foo}_{\mathsf{B},\mathsf{D}}$, because we have three constructors. In more detail, we have to define f on canonical terms of the form c_1, $(\mathsf{c}_2\ \mathsf{e})$, and $(\mathsf{c}_3\ \mathsf{a}_1\ \mathsf{a}_2\ \mathsf{m})$, as follows:

$$\mathsf{f}\ \mathsf{c}_1 :\equiv \text{ some constant}$$
$$\mathsf{f}\ (\mathsf{c}_2\ \mathsf{e}) :\equiv \text{ some function dependent on } \mathsf{e}$$
$$\mathsf{f}\ (\mathsf{c}_3\ \mathsf{a}_1\ \mathsf{a}_2\ \mathsf{m}) :\equiv \text{ some function dependent on } \mathsf{a}_1, \mathsf{a}_2, \text{ and } \mathsf{m}, \text{ and recursive calls } (\mathsf{f}\ \mathsf{a}_1) \text{ and } (\mathsf{f}\ \mathsf{a}_2)$$

Notice that the last equation depends on recursive calls $(\mathsf{f}\ \mathsf{a}_1)$ and $(\mathsf{f}\ \mathsf{a}_2)$, because terms a_1, a_2 have type $\mathsf{Foo}_{\mathsf{B},\mathsf{D}}$.

Now, let us apply this pattern to (-1)-truncations. Terms $|\mathsf{a}|$ are in canonical form. Therefore, the induction principle will have the clause:

$$\mathsf{f}\ |\mathsf{a}| :\equiv \text{ some function dependent on } \mathsf{a}$$

But $|\mathsf{a}|$ are not the only terms produced by the constructors, there are also the identity proofs $|\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|$ produced by the higher constructor! Therefore, the induction principle must include a clause stating how f behaves when applied on identity proofs of the form $|\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|$.

The application operators (Lemma 1.5.62 and Lemma 1.5.63) are used when we want to express that a function is applied on identity proofs.

In the case for (-1)-truncations, we have to use the dependent version of the application operator apd (Lemma 1.5.63), because the induction principle builds a dependent function. Hence, the induction principle will have the following clause:

$$\begin{aligned}\mathsf{apd}_{\mathsf{f}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}| = &\text{ some function dependent on } \mathsf{w},\ \mathsf{y},\\ &\text{and recursive calls } (\mathsf{f}\ \mathsf{w}) \text{ and } (\mathsf{f}\ \mathsf{y})\end{aligned} \tag{1.30}$$

Notice that recursive calls $(\mathsf{f}\ \mathsf{w})$ and $(\mathsf{f}\ \mathsf{y})$ must be included in (1.30) because w and y have type $\|\mathsf{A}\|$.

Name k the required function on the right side of (1.30). Since f must have type $\prod_{\mathsf{w}:\|\mathsf{A}\|} \mathsf{P}\ \mathsf{w}$, we require $\mathsf{f}\ \mathsf{w} : \mathsf{P}\ \mathsf{w}$ and $\mathsf{f}\ \mathsf{y} : \mathsf{P}\ \mathsf{y}$ for any $\mathsf{w}, \mathsf{y} : \|\mathsf{A}\|$. This means that the type of k must start like:

$$\prod_{\mathsf{w},\mathsf{y}:\|\mathsf{A}\|} \prod_{\mathsf{a}:\mathsf{P}\ \mathsf{w}} \prod_{\mathsf{b}:\mathsf{P}\ \mathsf{y}} \cdots$$

where $\ldots$ is a type we still need to determine.

Hence, (1.30) can be written as the following *candidate* equation:

$$\mathsf{apd}_{\mathsf{f}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}| = \mathsf{k}\ \mathsf{w}\ \mathsf{y}\ (\mathsf{f}\ \mathsf{w})\ (\mathsf{f}\ \mathsf{y}) \tag{1.31}$$

However, we still need to determine the output type for k. Observe that $\mathsf{apd}_{\mathsf{f}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|$ has type (see Lemma 1.5.63):

$$\mathsf{transport}^{\mathsf{P}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|\ (\mathsf{f}\ \mathsf{w}) = \mathsf{f}\ \mathsf{y}$$

This means that term $\mathsf{k}\ \mathsf{w}\ \mathsf{y}\ (\mathsf{f}\ \mathsf{w})\ (\mathsf{f}\ \mathsf{y})$ *must* have this type if we want (1.31) to be well-defined. Hence, the type of k is determined to be:

$$\prod_{\mathsf{w},\mathsf{y}:\|\mathsf{A}\|} \prod_{\mathsf{a}:\mathsf{P}\ \mathsf{w}} \prod_{\mathsf{b}:\mathsf{P}\ \mathsf{y}} \mathsf{transport}^{\mathsf{P}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|\ \mathsf{a} = \mathsf{b}$$

The induction principle for (-1)-truncations can be interpreted topologically, but we will not delve into this interpretation, since the operational explanation developed above will suffice for the purposes on this report.[62]

Notice that Equation (1.31) is *not* stated using definitional equality. The HoTT book explains this by arguing that judgmental equalities are part of the deductive system, and so, they should not depend on definitions made inside the type theory, as it is the case for the apd function.

The induction principle can be simplified, as the following corollary shows.

Corollary 1.11.3 (Simplified induction principle for (-1)-truncations)**.** *Let* $\mathsf{A} : \mathcal{U}_i$ *be a type and* $\mathsf{P} : \|\mathsf{A}\| \to \mathbf{Prop}_j$ *a family of propositions. If we can build a function:*

$$\mathsf{h} : \prod_{\mathsf{a}:\mathsf{A}} \mathsf{P}\ |\mathsf{a}|$$

then we can define a function:

$$\mathsf{f} : \prod_{\mathsf{w}:\|\mathsf{A}\|} \mathsf{P}\ \mathsf{w}$$

by the following equation (for any $\mathsf{a} : \mathsf{A}$*):*

$$\mathsf{f}\ |\mathsf{a}| :\equiv \mathsf{h}\ \mathsf{a}$$

[62] The reader can have a look at Chapter 6 in the HoTT book [20], where the homotopy interpretation is explored in detail for some HITs.

Proof. By hypothesis, we already have function h. Therefore, by the induction principle (Definition 1.11.2) it remains to prove that there is a function:

$$\mathsf{k}\colon \prod_{\mathsf{w},\mathsf{y}:\|\mathsf{A}\|} \prod_{\mathsf{a}:\mathsf{P}\ \mathsf{w}} \prod_{\mathsf{b}:\mathsf{P}\ \mathsf{y}} \mathsf{transport}^{\mathsf{P}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|\ \mathsf{a} = \mathsf{b}$$

So, let $\mathsf{w},\mathsf{y}\colon \|\mathsf{A}\|$, $\mathsf{a}\colon \mathsf{P}\ \mathsf{w}$, and $\mathsf{b}\colon \mathsf{P}\ \mathsf{y}$. By hypothesis, $\mathsf{P}\ \mathsf{y}$ is a mere proposition. Therefore, the following type is trivially inhabited:

$$\mathsf{transport}^{\mathsf{P}}\ |\mathsf{iden}\ \mathsf{w}\ \mathsf{y}|\ \mathsf{a} =_{(\mathsf{P}\ \mathsf{y})} \mathsf{b}$$

Hence, a function k exists.

□

When the simplified induction principle is instantiated with a constant family, we get a recursion principle.

Corollary 1.11.4 (Recursion principle for (-1)-truncations)**.** *Let* $\mathsf{A}\colon \mathcal{U}_i$ *be a type and* $\mathsf{D}\colon \mathbf{Prop}_j$ *a proposition. If we can build a function:*

$$\mathsf{h}\colon \mathsf{A} \to \mathsf{D}$$

then we can define a function:

$$\mathsf{f}\colon \|\mathsf{A}\| \to \mathsf{D}$$

by the following equation (for any $\mathsf{a}\colon \mathsf{A}$*):*

$$\mathsf{f}\ |\mathsf{a}| :\equiv \mathsf{h}\ \mathsf{a}$$

Proof. Apply Corollary 1.11.3 with the constant family:

$$\mathsf{P} :\equiv (\lambda \mathsf{w}.\ \mathsf{D})\colon \|\mathsf{A}\| \to \mathbf{Prop}_j$$

□

The following remark explains how (-1)-truncations allow the representation of classical logic operators (as understood in first-order classical logic). Also, the remark provides an intuitive explanation for the recursion principle.

Remark 1.11.5 (Logical interpretation of (-1)-truncations)**.** Classical existence of proofs can be modeled using (-1)-truncated types. Given a proof $\mathsf{p}\colon \|\mathsf{T}\|$, this can be interpreted as: "p is a proof to the fact that there is *some* proof for T".

To understand this statement, observe that the only way to construct terms in $\|\mathsf{T}\|$ is through one of its constructors. The higher constructor $|\mathsf{iden}|$ produces proofs for identities between terms *already* existing in $\|\mathsf{T}\|$, and the constructor $|\ |$ produces a term if provided with a term in T. In other words, if there are no terms in T, then no constructor for $\|\mathsf{T}\|$ will be applicable, producing as a result *no* terms in $\|\mathsf{T}\|$. Therefore, if we have $\mathsf{p}\colon \|\mathsf{T}\|$, then *some* term in T must exist (otherwise, the existence of p is impossible), though we cannot pinpoint which term it is.

To see why we cannot pinpoint the actual proof for T when there is a proof $\mathsf{p}\colon \|\mathsf{T}\|$, observe that p is identifiable with $|\mathsf{t}|$ for *every* $\mathsf{t}\colon \mathsf{T}$, since $\|\mathsf{T}\|$ is a mere proposition. This means that it is possible to be in a situation where we have two non-identifiable proofs for T (say $\mathsf{r},\mathsf{s}\colon \mathsf{T}$), but we will have $\mathsf{p} = |\mathsf{r}|$ and $\mathsf{p} = |\mathsf{s}|$. So, the question is: Which one is the proof for T? Is it term t or term s? We cannot arbitrarily choose t or s as the proof for T, because they are non-identifiable in T.

Since $\|\mathsf{T}\|$ means that T has some proof, this suggest we can obtain classical logic operators by (-1)-truncation. For example, if we (-1)-truncate a Σ-type:

$$\left\| \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w} \right\|$$

this can be interpreted as a classical existential, as in classical first-order logic:

$$\exists \mathsf{w}\colon \mathsf{A}.\ \mathsf{P}\ \mathsf{w}$$

Let us explain why it is a classical existential. In Remark 1.5.30, we explained that a sigma $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ is a constructive existential, in the sense that any proof for $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ is a pair (α,β), where α is the object claimed to

exist, and β is a proof that object α satisfies P. But, if we have a proof for $\left\|\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right\|$ this means that there is *some* pair that inhabits $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$. In other words, there is (in the classical sense) an object $\mathsf{w}:\mathsf{A}$ satisfying $\mathsf{P}\ \mathsf{w}$, i.e. we cannot point our finger to the object whose existence is claimed by the truncated sigma.

Similarly, if we (-1)-truncate a sum type:

$$\|\mathsf{A} + \mathsf{B}\|$$

this can be interpreted as a classical disjunction, as in classical first-order logic:

$$\mathsf{A} \vee \mathsf{B}$$

Again, to understand why this can be interpreted as such, in Remark 1.5.38 we explained that a sum type can be interpreted as a constructive disjunction, in the sense that a proof for $\mathsf{A} + \mathsf{B}$ always indicates the case that actually holds. However, if we have a proof for $\|\mathsf{A} + \mathsf{B}\|$ this means that there is *some* term that inhabits $\mathsf{A} + \mathsf{B}$. In other words, we know either A or B, but we no longer know which one holds, because we cannot pinpoint the proof for $\mathsf{A} + \mathsf{B}$.

With this logical interpretation for (-1)-truncations, the recursion principle (Corollary 1.11.4) now reads: "Let D be a mere proposition. To construct a proof for D assuming that A has *some* proof, it is enough to construct a proof for D *by assuming we have a concrete proof* for A". If we know there is *some* proof for A, and we also know that any proof for A can be transformed into a proof for D, then it will not matter the actual proof for A that gets *chosen* when constructing a proof for D, since D is a mere proposition.

In the previous paragraph, it may not be clear the significance of condition: "D is a mere proposition". To see why this condition is absolutely critical for the soundness of the recursion principle, let us reword our explanation in terms of functions.

The recursion principle is attempting to build a function $\mathsf{f}:\|\mathsf{A}\| \to \mathsf{D}$ by defining it on canonical terms of the form $|\mathsf{t}|$, for arbitrary $\mathsf{t}:\mathsf{A}$. Since *all* terms in $\|\mathsf{A}\|$ are identified, this is effectively saying that function f is *arbitrarily choosing a proof for* A *on its definition*, because it only knows that *some* proof for A exists.

When D is not a mere proposition, f is not well-defined, as we explain next. For any $\mathsf{t},\mathsf{r}:\mathsf{A}$, type $|\mathsf{t}| = |\mathsf{r}|$ is always inhabited. Therefore, by the application operator ap_{f}, type $\mathsf{f}\ |\mathsf{t}| = \mathsf{f}\ |\mathsf{r}|$ is also inhabited. But it may be the case that $\mathsf{f}\ |\mathsf{t}|$ and $\mathsf{f}\ |\mathsf{r}|$ are non-identifiable terms in D (since D is not a mere proposition), which will produce a contradiction. In other words, when D is not a mere proposition, it is not sound to arbitrarily choose a proof for A in the definition of f.

▲

To clarify the difference between classical operators and constructive operators, let us compare the so-called "Choice Theorem" and the "Axiom of Choice".

Theorem 1.11.6 (Choice Theorem)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *be a type,* $\mathsf{B}:\mathsf{A}\to\mathcal{U}_j$ *a type family, and* $\mathsf{P}:\prod_{\mathsf{w}:\mathsf{A}}((\mathsf{B}\ \mathsf{w})\to\mathcal{U}_k)$ *another type family.*

The following type is inhabited:

$$\left(\prod_{\mathsf{w}:\mathsf{A}} \sum_{\mathsf{a}:\mathsf{B}\ \mathsf{w}} \mathsf{P}\ \mathsf{w}\ \mathsf{a}\right) \to \left(\sum_{\mathsf{g}:\prod_{\mathsf{y}:\mathsf{A}} \mathsf{B}\ \mathsf{y}} \prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\ (\mathsf{g}\ \mathsf{w})\right)$$

Proof. Suppose $\mathsf{f}:\prod_{\mathsf{w}:\mathsf{A}} \sum_{\mathsf{a}:\mathsf{B}\ \mathsf{w}} \mathsf{P}\ \mathsf{w}\ \mathsf{a}$. Define:

$$\mathsf{g} :\equiv (\lambda \mathsf{y}:\mathsf{A}.\ \mathsf{pr}_1\ (\mathsf{f}\ \mathsf{y})):\prod_{\mathsf{y}:\mathsf{A}} \mathsf{B}\ \mathsf{y}$$

It remains to prove that type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\ (\mathsf{g}\ \mathsf{w})$ is inhabited.

Let $\mathsf{w}:\mathsf{A}$. But we have $\mathsf{pr}_2\ (\mathsf{f}\ \mathsf{w}):\mathsf{P}\ \mathsf{w}\ (\mathsf{pr}_1\ (\mathsf{f}\ \mathsf{w}))$, or equivalently, by definition of g:

$$\mathsf{pr}_2\ (\mathsf{f}\ \mathsf{w}):\mathsf{P}\ \mathsf{w}\ (\mathsf{g}\ \mathsf{w})$$

□

If we logically interpret the Choice Theorem, it states: "*if* for every w in A, there is $\mathsf{a}:\mathsf{B}\,\mathsf{w}$ such that $(\mathsf{P}\,\mathsf{w}\,\mathsf{a})$ holds, *then* we can construct a *choice function* g, which picks for every $\mathsf{y}:\mathsf{A}$ an element of $(\mathsf{B}\,\mathsf{y})$, so that $\mathsf{P}\,\mathsf{w}\,(\mathsf{g}\,\mathsf{w})$ holds for any $\mathsf{w}:\mathsf{A}$".

This reads exactly like the axiom of choice in standard mathematics, but in here, it is a theorem instead of an axiom. How can we explain this? The reason is that all Σ-types in the Choice Theorem are constructive existentials. Σ-types are picking an specific object when they claim "there is...". Hence, building the choice function g amounts to taking the object that the Σ-type already chose for us!

Therefore, if we want to encode the classical intuition behind the axiom of choice, we have to (-1)-truncate all Σ-types in the Choice Theorem. To make the statement closer to the axiom of choice, we will word it in terms of sets and propositions as follows.

Given a set $\mathsf{A}:\mathbf{Set}_i$, a family of sets $\mathsf{B}:\mathsf{A}\to\mathbf{Set}_j$, and a binary predicate $\mathsf{P}:\prod_{\mathsf{w}:\mathsf{A}}((\mathsf{B}\,\mathsf{w})\to\mathbf{Prop}_k)$, the following type encodes the so-called "Axiom of Choice":

$$\left(\prod_{\mathsf{w}:\mathsf{A}}\left\|\sum_{\mathsf{a}:\mathsf{B}\,\mathsf{w}}\mathsf{P}\,\mathsf{w}\,\mathsf{a}\right\|\right)\to\left\|\sum_{\mathsf{g}:\prod_{\mathsf{y}:\mathsf{A}}\mathsf{B}\,\mathsf{y}}\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}\,(\mathsf{g}\,\mathsf{w})\right\| \tag{1.32}$$

This reads: "*if* for every w in A, there is (in the classical sense) $\mathsf{a}:\mathsf{B}\,\mathsf{w}$ such that $(\mathsf{P}\,\mathsf{w}\,\mathsf{a})$ holds, *then* there is (in the classical sense) a *choice function* g, which picks for every $\mathsf{y}:\mathsf{A}$ an element of $(\mathsf{B}\,\mathsf{y})$, so that $\mathsf{P}\,\mathsf{w}\,(\mathsf{g}\,\mathsf{w})$ holds for every $\mathsf{w}:\mathsf{A}$".

Type (1.32) cannot be proved to be inhabited in HoTT [20]. It has to be added as an axiom if we want to use it. *On this report we will not make use of the axiom of choice.*

The following explanation provides some intuition on why the axiom of choice cannot be proved to be inhabited. If we attempt to construct the choice function g, we cannot pick an object from the sigma in the hypothesis, because the truncated sigma is no longer making a choice for us (i.e. we only know that *some* object exists, but there is no "procedure" for choosing it). Hence, the conclusion on type (1.32) is stating that *some* choice function exists, even though there is no way to define it constructively.

However, there are special cases where we can extract a choice function. For example, when there is exactly one object, we can choose it constructively. This is the so-called "Principle of unique choice". To prove this principle, we need to prove the following lemma, which states that (-1)-truncating a proposition has no effect on it.

Lemma 1.11.7. *Let* $\mathsf{A}:\mathbf{Prop}_i$ *be a proposition. Then* $\mathsf{A}\simeq\|\mathsf{A}\|$ *is inhabited.*

Proof. Since A and $\|\mathsf{A}\|$ are mere propositions, by Lemma 1.8.8 it is enough to prove that they are logically equivalent.

For the first function, we already have the (-1)-truncation constructor $|\,|:\mathsf{A}\to\|\mathsf{A}\|$. It remains to define the second function.

By the recursion principle for (-1)-truncations (Corollary 1.11.4), to build a function $\|\mathsf{A}\|\to\mathsf{A}$ it is enough to build a function $\mathsf{A}\to\mathsf{A}$, since A is a mere proposition. The identity function $\mathsf{id}_{\mathsf{A}}:\mathsf{A}\to\mathsf{A}$ will suffice.

□

And now the principle, which is a consequence of the previous lemma.

Lemma 1.11.8 (Principle of unique choice)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *be a type and* $\mathsf{P}:\mathsf{A}\to\mathcal{U}_j$ *a type family.*

If the following types are inhabited:

$$\prod_{\mathsf{w}:\mathsf{A}}\mathsf{IsProp}\,(\mathsf{P}\,\mathsf{w})$$

$$\prod_{\mathsf{w}:\mathsf{A}}\|\mathsf{P}\,\mathsf{w}\|$$

then the following type is inhabited:

$$\prod_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}$$

Proof. Let $\mathsf{w} : \mathsf{A}$. We want to prove that $\mathsf{P}\ \mathsf{w}$ is inhabited.

By the first hypothesis, $\mathsf{P}\ \mathsf{w}$ is a mere proposition. Therefore, by Lemma 1.11.7, $\mathsf{P}\ \mathsf{w}$ and $\|\mathsf{P}\ \mathsf{w}\|$ are equivalent. So, we have an equivalence $\mathsf{e} : (\mathsf{P}\ \mathsf{w}) \to \|\mathsf{P}\ \mathsf{w}\|$.

By the second hypothesis, we have a proof $\mathsf{h} : \|\mathsf{P}\ \mathsf{w}\|$. Hence:

$$\mathsf{e}^{-1}\ \mathsf{h} : \mathsf{P}\ \mathsf{w}$$

□

In other words, given $\mathsf{w} : \mathsf{A}$, if there is at most one proof for $\mathsf{P}\ \mathsf{w}$ (i.e. $\mathsf{IsProp}\ (\mathsf{P}\ \mathsf{w})$ is inhabited), *and* there is (in the classical sense) a proof for $\mathsf{P}\ \mathsf{w}$ (i.e. $\|\mathsf{P}\ \mathsf{w}\|$ is inhabited), then there is *exactly one* proof for $\mathsf{P}\ \mathsf{w}$, which we can certainly choose. Therefore, we can define a function $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ constructively.

Surjective functions are another classical concept that can be defined using (-1)-truncations.

Definition 1.11.9 (Surjective function)**.** Let $\mathsf{A} : \mathcal{U}_i$ and $\mathsf{B} : \mathcal{U}_j$ be types, and $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ a function. We define type:

$$\mathsf{IsSurjection}\ \mathsf{f} :\equiv \prod_{\mathsf{b}:\mathsf{B}} \left\| \sum_{\mathsf{a}:\mathsf{A}} \mathsf{f}\ \mathsf{a} = \mathsf{b} \right\|$$

If $\mathsf{IsSurjection}\ \mathsf{f}$ is inhabited, we say that f is a *surjection*, or that f is a *surjective* function.

▲

In other words, f is surjective if every element on its codomain is the image of some (in the classical sense) element on its domain.

The identity function is trivially surjective.

Lemma 1.11.10. *Let $\mathsf{A} : \mathcal{U}_i$ be a type. Then id_{A} is surjective.*

Proof. Let $\mathsf{a} : \mathsf{A}$. Then $|(\mathsf{a}, \mathsf{refl}_{\mathsf{A}}\ \mathsf{a})| : \|\sum_{\mathsf{b}:\mathsf{A}} \mathsf{id}_{\mathsf{A}}\ \mathsf{b} = \mathsf{a}\|$. □

Surjective functions compose as expected.

Lemma 1.11.11. *Let $\mathsf{A} : \mathcal{U}_i$, $\mathsf{B} : \mathcal{U}_j$ and $\mathsf{C} : \mathcal{U}_k$ be types. Let $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ and $\mathsf{g} : \mathsf{B} \to \mathsf{C}$ be functions.*

If f and g are surjective, then $\mathsf{g} \circ \mathsf{f}$ is surjective.

Proof. Let $\mathsf{c} : \mathsf{C}$. Since g is surjective, type $\|\sum_{\mathsf{b}:\mathsf{B}} \mathsf{g}\ \mathsf{b} = \mathsf{c}\|$ is inhabited. Therefore, we will be done if we are able to prove:

$$\left\| \sum_{\mathsf{b}:\mathsf{B}} \mathsf{g}\ \mathsf{b} = \mathsf{c} \right\| \to \left\| \sum_{\mathsf{a}:\mathsf{A}} \mathsf{g}\ (\mathsf{f}\ \mathsf{a}) = \mathsf{c} \right\|$$

By (-1)-recursion (Corollary 1.11.4), it is enough to prove:

$$\left(\sum_{\mathsf{b}:\mathsf{B}} \mathsf{g}\ \mathsf{b} = \mathsf{c} \right) \to \left\| \sum_{\mathsf{a}:\mathsf{A}} \mathsf{g}\ (\mathsf{f}\ \mathsf{a}) = \mathsf{c} \right\|$$

which in turn, by Σ-recursion (Lemma 1.5.25), it is enough to prove:

$$\prod_{\mathsf{b}:\mathsf{B}} \left((\mathsf{g}\ \mathsf{b} = \mathsf{c}) \to \left\| \sum_{\mathsf{a}:\mathsf{A}} \mathsf{g}\ (\mathsf{f}\ \mathsf{a}) = \mathsf{c} \right\| \right)$$

So, let $\mathsf{b} : \mathsf{B}$ with $\mathsf{p}_1 : \mathsf{g}\ \mathsf{b} = \mathsf{c}$. Since f is surjective, type $\|\sum_{\mathsf{a}:\mathsf{A}} \mathsf{f}\ \mathsf{a} = \mathsf{b}\|$ is inhabited. Therefore, we will be done if we are able to prove:

$$\left\| \sum_{\mathsf{a}:\mathsf{A}} \mathsf{f}\ \mathsf{a} = \mathsf{b} \right\| \to \left\| \sum_{\mathsf{a}:\mathsf{A}} \mathsf{g}\ (\mathsf{f}\ \mathsf{a}) = \mathsf{c} \right\|$$

By (-1)-recursion and Σ-recursion, it is enough to prove:

$$\prod_{a:A}\left((f\ a = b) \to \left\|\sum_{a:A} g\ (f\ a) = c\right\|\right)$$

So, let $a : A$ with $p_2 : f\ a = b$. Then, we have $p_3 :\equiv ap_g\ p_2 : g\ (f\ a) = g\ b$. Hence:

$$|(a, p_3 \centerdot p_1)| : \left\|\sum_{a:A} g\ (f\ a) = c\right\|$$

□

1.11.2 0-truncations

If (-1)-truncating a type produces a mere proposition, 0-truncating a type will produce a set.[63]

Definition 1.11.12 (Formation and introduction rules for 0-truncations)**.** Let $A : \mathcal{U}_i$ be a type. There is a type at level i, denoted $\|A\|_0$, and called the 0-*truncation* of A.

Type $\|A\|_0$ has the following two constructors:

- Term constructor. $|\ |_0 : A \to \|A\|_0$.
  Instead of $|\ |_0\ a$, we will write $|a|_0$.
- Higher constructor. $|\mathsf{iden}|_0 : \prod_{w,y:\|A\|_0} \prod_{p,q:w=y} p = q$.
  Instead of $|\mathsf{iden}|_0\ w\ y\ p\ q$, we will write $|\mathsf{iden}\ p\ q|_0$, where w and y are left implicit.

▲

Constructor $|\ |_0$ copies all terms in A, while constructor $|\mathsf{iden}|_0$ introduces the proof $|\mathsf{iden}\ p\ q|_0 : p = q$ for any two identity proofs $p, q : w = y$ in $\|A\|_0$. In other words, constructor $|\mathsf{iden}|_0$ is expressing that $\|A\|_0$ is a set.

Although it is possible to state a general version of the induction principle for 0-truncations (as we did for (-1)-truncations), we will state the simplified version,[64] as this is the version we will use on this report.

Definition 1.11.13 (Induction principle for 0-truncations)**.** Let $A : \mathcal{U}_i$ be a type and $P : \|A\|_0 \to \mathbf{Set}_j$ a family of sets. If we can build a function:

$$h : \prod_{a:A} P\ |a|_0$$

then we can define a function:

$$f : \prod_{w:\|A\|_0} P\ w$$

by the following equation (for any $a : A$):

$$f\ |a|_0 :\equiv h\ a$$

▲

We also have a recursion principle as a consequence.

Corollary 1.11.14 (Recursion principle for 0-truncations)**.** *Let $A : \mathcal{U}_i$ be a type and $D : \mathbf{Set}_j$ a set. If we can build a function:*

$$h : A \to D$$

then we can define a function:

$$f : \|A\|_0 \to D$$

by the following equation (for any $a : A$):

$$f\ |a|_0 :\equiv h\ a$$

Proof. Apply Definition 1.11.13 with the constant family:

$$P :\equiv (\lambda w.\ D) : \|A\|_0 \to \mathbf{Set}_j$$

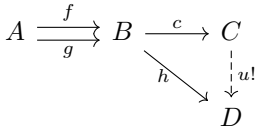
□

[63] Sets are also called 0-types, see Definition 1.8.9.

[64] The reader can find the more general version at Section 6.9 in the HoTT book [20].

1.11.3 Coequalizers

In category theory [5], the coequalizer of two arrows $f, g : A \to B$ is an object C together with an arrow $c : B \to C$ such that $c \circ f = c \circ g$. Moreover, the object C and arrow c must satisfy the following universal property: for any arrow $h : B \to D$ satisfying $h \circ f = h \circ g$, there is a *unique* arrow $u : C \to D$ making the following diagram commute:



The coequalizer C can be modeled as a HIT, where arrow c acts as its constructor, and condition $c \circ f = c \circ g$ as its higher constructor. In fact, the universal property for coequalizers will correspond to the induction/recursion principles, since these principles construct functions out of C.

Definition 1.11.15 (Formation and introduction rules for coequalizers)**.** Let $\mathsf{A} : \mathcal{U}_i$ and $\mathsf{B} : \mathcal{U}_j$ be types. Let $\mathsf{f}, \mathsf{g} : \mathsf{A} \to \mathsf{B}$ be two functions.

There is a type at level $\max\{i, j\}$, denoted $\mathsf{Coeq\ f\ g}$, and called the *coequalizer* for f and g.

Type $\mathsf{Coeq\ f\ g}$ has the following two constructors:

- Term constructor. $\mathsf{coeq} : \mathsf{B} \to \mathsf{Coeq\ f\ g}$.
- Higher constructor. $\mathsf{coeqiden} : \prod_{\mathsf{a:A}} \mathsf{coeq\ (f\ a)} = \mathsf{coeq\ (g\ a)}$.

▲

Type $\mathsf{Coeq\ f\ g}$ is subject to the following induction principle.

Definition 1.11.16 (Induction principle for coequalizers)**.** Let $\mathsf{A} : \mathcal{U}_i$, $\mathsf{B} : \mathcal{U}_j$ be types, $\mathsf{f}, \mathsf{g} : \mathsf{A} \to \mathsf{B}$ two functions, and $\mathsf{P} : (\mathsf{Coeq\ f\ g}) \to \mathcal{U}_k$ a type family.

If we can build two functions:

$$\mathsf{h} : \prod_{\mathsf{b:B}} \mathsf{P\ (coeq\ b)}$$

$$\mathsf{k} : \prod_{\mathsf{a:A}} \mathsf{transport^P\ (coeqiden\ a)\ (h\ (f\ a)) = h\ (g\ a)}$$

then we can define a function:

$$\mathsf{u} : \prod_{\mathsf{w:Coeq\ f\ g}} \mathsf{P\ w}$$

by the following equations:[65]

$$\mathsf{u\ (coeq\ b)} :\equiv \mathsf{h\ b}$$

$$\mathsf{apd_u\ (coeqiden\ a)} = \mathsf{k\ a}$$

for any $\mathsf{b} : \mathsf{B}$ and $\mathsf{a} : \mathsf{A}$.

▲

Notice that the equations defining u conform to the "general pattern":

$$\begin{aligned} \mathsf{u\ (coeq\ b)} &:\equiv \text{ some function dependent on } \mathsf{b} \\ \mathsf{apd_u\ (coeqiden\ a)} &= \text{ some function dependent on } \mathsf{a} \end{aligned}$$

where the type of functions h and k (i.e. the required functions on the right side of the equations) is determined by the terms on the left side of the equations.

Instead of this general induction principle, we will use the following simplified version.

[65] Notice that the second equation is well-typed, since $\mathsf{apd_u\ (coeqiden\ a) : transport^P\ (coeqiden\ a)\ (u\ (coeq\ (f\ a))) = u\ (coeq\ (g\ a))}$, which simplifies to $\mathsf{apd_u\ (coeqiden\ a) : transport^P\ (coeqiden\ a)\ (h\ (f\ a)) = h\ (g\ a)}$ by the first equation.

Lemma 1.11.17 (Simplified induction principle for coequalizers)**.** *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$ *be types,* $\mathsf{f},\mathsf{g}:\mathsf{A}\to\mathsf{B}$ *two functions, and* $\mathsf{P}:(\mathsf{Coeq}\ \mathsf{f}\ \mathsf{g})\to\mathbf{Prop}_k$ *a family of propositions.*

If the following type is inhabited:

$$\prod_{\mathsf{b}:\mathsf{B}}\mathsf{P}\ (\mathsf{coeq}\ \mathsf{b})$$

then the following type is inhabited:

$$\prod_{\mathsf{w}:\mathsf{Coeq}\ \mathsf{f}\ \mathsf{g}}\mathsf{P}\ \mathsf{w}$$

Proof. We are given a function $\mathsf{h}:\prod_{\mathsf{b}:\mathsf{B}}\mathsf{P}\ (\mathsf{coeq}\ \mathsf{b})$ by hypothesis. Therefore, by the induction principle, it is enough to prove that the following type is inhabited:

$$\prod_{\mathsf{a}:\mathsf{A}}\mathsf{transport}^{\mathsf{P}}\ (\mathsf{coeqiden}\ \mathsf{a})\ (\mathsf{h}\ (\mathsf{f}\ \mathsf{a}))=\mathsf{h}\ (\mathsf{g}\ \mathsf{a})$$

Let $\mathsf{a}:\mathsf{A}$. Since $\mathsf{P}\ (\mathsf{coeq}\ (\mathsf{g}\ \mathsf{a}))$ is a mere proposition, the identity type is trivially inhabited.

□

We also have a (simplified) recursion principle.

Lemma 1.11.18 (Recursion principle for coequalizers)**.** *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$ *be types,* $\mathsf{f},\mathsf{g}:\mathsf{A}\to\mathsf{B}$ *two functions, and* $\mathsf{D}:\mathcal{U}_k$ *a type.*

If we can build a function:

$$\mathsf{h}:\mathsf{B}\to\mathsf{D}$$

such that the following type is inhabited:

$$\prod_{\mathsf{a}:\mathsf{A}}\mathsf{h}\ (\mathsf{f}\ \mathsf{a})=\mathsf{h}\ (\mathsf{g}\ \mathsf{a})$$

then there is a function:

$$\mathsf{u}:(\mathsf{Coeq}\ \mathsf{f}\ \mathsf{g})\to\mathsf{D}$$

satisfying (for every $\mathsf{b}:\mathsf{B}$*):*

$$\mathsf{u}\ (\mathsf{coeq}\ \mathsf{b})\equiv\mathsf{h}\ \mathsf{b}$$

Proof. We are given two functions:

$$\mathsf{h}:\mathsf{B}\to\mathsf{D}$$
$$\mathsf{k}:\prod_{\mathsf{a}:\mathsf{A}}\mathsf{h}\ (\mathsf{f}\ \mathsf{a})=\mathsf{h}\ (\mathsf{g}\ \mathsf{a})$$

Define the constant family:

$$\mathsf{P}:\equiv(\lambda\mathsf{w}.\ \mathsf{D}):(\mathsf{Coeq}\ \mathsf{f}\ \mathsf{g})\to\mathcal{U}_k$$

Then, by the induction principle (Definition 1.11.16), it remains to prove:

$$\prod_{\mathsf{a}:\mathsf{A}}\mathsf{transport}^{\mathsf{P}}\ (\mathsf{coeqiden}\ \mathsf{a})\ (\mathsf{h}\ (\mathsf{f}\ \mathsf{a}))=\mathsf{h}\ (\mathsf{g}\ \mathsf{a})$$

So, let $\mathsf{a}:\mathsf{A}$. By Lemma 1.6.6, there is a proof:

$$\mathsf{q}_1:\mathsf{transport}^{\mathsf{P}}\ (\mathsf{coeqiden}\ \mathsf{a})\ (\mathsf{h}\ (\mathsf{f}\ \mathsf{a}))=\mathsf{h}\ (\mathsf{f}\ \mathsf{a})$$

which means:

$$\mathsf{q}_1\cdot(\mathsf{k}\ \mathsf{a}):\mathsf{transport}^{\mathsf{P}}\ (\mathsf{coeqiden}\ \mathsf{a})\ (\mathsf{h}\ (\mathsf{f}\ \mathsf{a}))=\mathsf{h}\ (\mathsf{g}\ \mathsf{a})$$

□

Notice that the recursion principle reads like the universal property for coequalizers. In fact, it can be proved that the function produced by the induction principle is unique, but we omit the proof.

The coequalizer constructor coeq is always surjective, as the following lemma shows.

Lemma 1.11.19. *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$ *be types, and* $\mathsf{f},\mathsf{g}:\mathsf{A}\to\mathsf{B}$ *two functions. The constructor* $\mathsf{coeq}:\mathsf{B}\to(\mathsf{Coeq}\ \mathsf{f}\ \mathsf{g})$ *is surjective.*

Proof. To prove:

$$\prod_{\mathsf{w}:\mathsf{Coeq}\ \mathsf{f}\ \mathsf{g}} \left\| \sum_{\mathsf{b}:\mathsf{B}} \mathsf{coeq}\ \mathsf{b} = \mathsf{w} \right\|$$

it is enough to prove:

$$\prod_{\mathsf{a}:\mathsf{B}} \left\| \sum_{\mathsf{b}:\mathsf{B}} \mathsf{coeq}\ \mathsf{b} = \mathsf{coeq}\ \mathsf{a} \right\|$$

by the induction principle (Lemma 1.11.17).

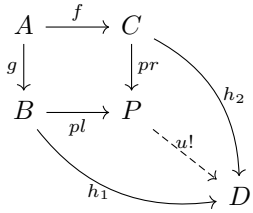
Let $\mathsf{a}:\mathsf{B}$. Therefore:

$$\left|(\mathsf{a}, \mathsf{refl}_{(\mathsf{Coeq}\ \mathsf{f}\ \mathsf{g})}\ (\mathsf{coeq}\ \mathsf{a}))\right| : \left\| \sum_{\mathsf{b}:\mathsf{B}} \mathsf{coeq}\ \mathsf{b} = \mathsf{coeq}\ \mathsf{a} \right\|$$

□

1.11.4 Pushouts

In category theory, the pushout of two arrows $f : A \to C$ and $g : A \to B$ is an object P together with two arrows $pl : B \to P$ and $pr : C \to P$ such that $pl \circ g = pr \circ f$. Moreover, the object P and arrows pl, pr must satisfy the following universal property: for any two arrows $h_1 : B \to D$ and $h_2 : C \to D$ satisfying $h_1 \circ g = h_2 \circ f$, there is a *unique* arrow $u : P \to D$ making the following diagram commute:



It is possible to model pushout P as a HIT, with arrows pl and pr as its constructors, and condition $pl \circ g = pr \circ f$ is its higher constructor. However, it is also known [5] that pushout P can be defined as the coequalizer for:

$$A \underset{inr \circ f}{\overset{inl \circ g}{\rightrightarrows}} B \sqcup C$$

where $B \sqcup C$ is the coproduct of B and C, and $inl : B \to B \sqcup C$ and $inr : C \to B \sqcup C$ are the coproduct injection arrows.

We will define pushouts in terms of coequalizers.

Lemma 1.11.20 (Formation and introduction for pushouts)**.** *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$, *and* $\mathsf{C}:\mathcal{U}_k$ *be types. Let* $\mathsf{g}:\mathsf{A}\to\mathsf{B}$ *and* $\mathsf{f}:\mathsf{A}\to\mathsf{C}$ *be two functions.*

There is a type at level $\max\{i,j,k\}$*, denoted* $\mathsf{Push}\ \mathsf{g}\ \mathsf{f}$*, and called the* pushout *for* g *and* f*.*

Type $\mathsf{Push}\ \mathsf{g}\ \mathsf{f}$ *has the following three constructors:*

- *Term constructor.* $\mathsf{pushl}:\mathsf{B}\to\mathsf{Push}\ \mathsf{g}\ \mathsf{f}$.
- *Term constructor.* $\mathsf{pushr}:\mathsf{C}\to\mathsf{Push}\ \mathsf{g}\ \mathsf{f}$.
- *Higher constructor.* $\mathsf{pushiden}:\prod_{\mathsf{a}:\mathsf{A}}\mathsf{pushl}\ (\mathsf{g}\ \mathsf{a}) = \mathsf{pushr}\ (\mathsf{f}\ \mathsf{a})$.

Proof. Define:

$$\mathsf{Push}\ \mathsf{g}\ \mathsf{f} :\equiv \mathsf{Coeq}\ (\mathsf{inl}\circ\mathsf{g})\ (\mathsf{inr}\circ\mathsf{f})$$

where $\mathsf{inl}:\mathsf{B}\to\mathsf{B}+\mathsf{C}$ and $\mathsf{inr}:\mathsf{C}\to\mathsf{B}+\mathsf{C}$.

Notice that $\mathsf{Push}\ \mathsf{g}\ \mathsf{f}$ is at level $\max\{i,j,k\}$ since $\mathsf{Coeq}\ (\mathsf{inl}\circ\mathsf{g})\ (\mathsf{inr}\circ\mathsf{f})$ must be at level $\max\{i,\max\{j,k\}\}$.

Now, define:

$$\mathsf{pushl} :\equiv (\lambda \mathsf{b}.\ \mathsf{coeq}\ (\mathsf{inl}\ \mathsf{b})) : \mathsf{B} \to \mathsf{Push}\ \mathsf{g}\ \mathsf{f}$$
$$\mathsf{pushr} :\equiv (\lambda \mathsf{c}.\ \mathsf{coeq}\ (\mathsf{inr}\ \mathsf{c})) : \mathsf{C} \to \mathsf{Push}\ \mathsf{g}\ \mathsf{f}$$
$$\mathsf{pushiden} :\equiv (\lambda \mathsf{a}.\ \mathsf{coeqiden}\ \mathsf{a}) : \prod_{\mathsf{a}:\mathsf{A}} \mathsf{pushl}\ (\mathsf{g}\ \mathsf{a}) = \mathsf{pushr}\ (\mathsf{f}\ \mathsf{a})$$

Notice that pushiden is well-typed since:

$$\mathsf{coeqiden}\ \mathsf{a} : \mathsf{coeq}\ (\mathsf{inl}\ (\mathsf{g}\ \mathsf{a})) = \mathsf{coeq}\ (\mathsf{inr}\ (\mathsf{f}\ \mathsf{a}))$$

or equivalently:

$$\mathsf{coeqiden}\ \mathsf{a} : \mathsf{pushl}\ (\mathsf{g}\ \mathsf{a}) = \mathsf{pushr}\ (\mathsf{f}\ \mathsf{a})$$

by definition of pushl and pushr.

□

We have the following (simplified) induction and recursion principles for pushouts.

Lemma 1.11.21 (Induction principle for pushouts)**.** *Let* $\mathsf{A} : \mathcal{U}_i$*,* $\mathsf{B} : \mathcal{U}_j$*, and* $\mathsf{C} : \mathcal{U}_k$ *be types. Let* $\mathsf{g} : \mathsf{A} \to \mathsf{B}$ *and* $\mathsf{f} : \mathsf{A} \to \mathsf{C}$ *be two functions, and* $\mathsf{P} : (\mathsf{Push}\ \mathsf{g}\ \mathsf{f}) \to \mathbf{Prop}_m$ *a family of propositions.*

If the following types are inhabited:

$$\prod_{\mathsf{b}:\mathsf{B}} \mathsf{P}\ (\mathsf{pushl}\ \mathsf{b})$$
$$\prod_{\mathsf{c}:\mathsf{C}} \mathsf{P}\ (\mathsf{pushr}\ \mathsf{c})$$

then the following type is inhabited:

$$\prod_{\mathsf{w}:\mathsf{Push}\ \mathsf{g}\ \mathsf{f}} \mathsf{P}\ \mathsf{w}$$

Proof. We are trying to prove:

$$\prod_{\mathsf{w}:\mathsf{Coeq}\ (\mathsf{inl} \circ \mathsf{g})\ (\mathsf{inr} \circ \mathsf{f})} \mathsf{P}\ \mathsf{w}$$

So, by Lemma 1.11.17, it is enough to prove:

$$\prod_{\mathsf{w}:\mathsf{B}+\mathsf{C}} \mathsf{P}\ (\mathsf{coeq}\ \mathsf{w})$$

but by the induction principle for sum types (Lemma 1.5.35), it is enough to prove:

$$\prod_{\mathsf{b}:\mathsf{B}} \mathsf{P}\ (\mathsf{coeq}\ (\mathsf{inl}\ \mathsf{b}))$$
$$\prod_{\mathsf{c}:\mathsf{C}} \mathsf{P}\ (\mathsf{coeq}\ (\mathsf{inr}\ \mathsf{c}))$$

or equivalently, by definition of pushl and pushr:

$$\prod_{\mathsf{b}:\mathsf{B}} \mathsf{P}\ (\mathsf{pushl}\ \mathsf{b})$$
$$\prod_{\mathsf{c}:\mathsf{C}} \mathsf{P}\ (\mathsf{pushr}\ \mathsf{c})$$

which are inhabited by hypothesis.

□

Lemma 1.11.22 (Recursion principle for pushouts)**.** *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$, *and* $\mathsf{C}:\mathcal{U}_k$ *be types. Let* $\mathsf{g}:\mathsf{A}\to\mathsf{B}$ *and* $\mathsf{f}:\mathsf{A}\to\mathsf{C}$ *be two functions, and* $\mathsf{D}:\mathcal{U}_m$ *a type.*

If there are two functions:

$$\mathsf{h}_1:\mathsf{B}\to\mathsf{D}$$
$$\mathsf{h}_2:\mathsf{C}\to\mathsf{D}$$

such that the following type is inhabited:

$$\prod_{\mathsf{a}:\mathsf{A}}\mathsf{h}_1\ (\mathsf{g}\ \mathsf{a})=\mathsf{h}_2\ (\mathsf{f}\ \mathsf{a})$$

then there is a function:

$$\mathsf{u}:(\mathsf{Push}\ \mathsf{g}\ \mathsf{f})\to\mathsf{D}$$

satisfying:

$$\mathsf{u}\ (\mathsf{pushl}\ \mathsf{b})\equiv\mathsf{h}_1\ \mathsf{b}$$
$$\mathsf{u}\ (\mathsf{pushr}\ \mathsf{c})\equiv\mathsf{h}_2\ \mathsf{c}$$

for any $\mathsf{b}:\mathsf{B}$ *and* $\mathsf{c}:\mathsf{C}$.

Proof. We are trying to construct a function $\mathsf{u}:(\mathsf{Coeq}\ (\mathsf{inl}\circ\mathsf{g})\ (\mathsf{inr}\circ\mathsf{f}))\to\mathsf{D}$. So, by Lemma 1.11.18, it is enough to construct a function $\mathsf{h}:(\mathsf{B}+\mathsf{C})\to\mathsf{D}$ satisfying $\prod_{\mathsf{a}:\mathsf{A}}\mathsf{h}\ (\mathsf{inl}\ (\mathsf{g}\ \mathsf{a}))=\mathsf{h}\ (\mathsf{inr}\ (\mathsf{f}\ \mathsf{a}))$.

By the recursion principle for sum types, we can define h by cases as (for arbitrary $\mathsf{b}:\mathsf{B}$ and $\mathsf{c}:\mathsf{C}$):

$$\mathsf{h}\ (\mathsf{inl}\ \mathsf{b}):\equiv\mathsf{h}_1\ \mathsf{b}$$
$$\mathsf{h}\ (\mathsf{inr}\ \mathsf{c}):\equiv\mathsf{h}_2\ \mathsf{c}$$

With this definition, $\prod_{\mathsf{a}:\mathsf{A}}\mathsf{h}\ (\mathsf{inl}\ (\mathsf{g}\ \mathsf{a}))=\mathsf{h}\ (\mathsf{inr}\ (\mathsf{f}\ \mathsf{a}))$ simplifies to:

$$\prod_{\mathsf{a}:\mathsf{A}}\mathsf{h}_1\ (\mathsf{g}\ \mathsf{a})=\mathsf{h}_2\ (\mathsf{f}\ \mathsf{a})$$

which is inhabited by hypothesis.

To finish the proof, we have by Lemma 1.11.18 and our definitions:

$$\mathsf{u}\ (\mathsf{pushl}\ \mathsf{b})\equiv\mathsf{u}\ (\mathsf{coeq}\ (\mathsf{inl}\ \mathsf{b}))\equiv\mathsf{h}\ (\mathsf{inl}\ \mathsf{b})\equiv\mathsf{h}_1\ \mathsf{b}$$
$$\mathsf{u}\ (\mathsf{pushr}\ \mathsf{c})\equiv\mathsf{u}\ (\mathsf{coeq}\ (\mathsf{inr}\ \mathsf{c}))\equiv\mathsf{h}\ (\mathsf{inr}\ \mathsf{c})\equiv\mathsf{h}_2\ \mathsf{c}$$

□

Notice that the recursion principle reads like the universal property for pushouts. Also, we will use the following property regarding the pushout constructors.

Lemma 1.11.23. *Let* $\mathsf{A}:\mathcal{U}_i$, $\mathsf{B}:\mathcal{U}_j$, *and* $\mathsf{C}:\mathcal{U}_k$ *be types. Let* $\mathsf{g}:\mathsf{A}\to\mathsf{B}$ *and* $\mathsf{f}:\mathsf{A}\to\mathsf{C}$ *be two functions.*

- *If* g *is surjective, then* $\mathsf{pushr}:\mathsf{C}\to\mathsf{Push}\ \mathsf{g}\ \mathsf{f}$ *is surjective.*
- *If* f *is surjective, then* $\mathsf{pushl}:\mathsf{B}\to\mathsf{Push}\ \mathsf{g}\ \mathsf{f}$ *is surjective.*

Proof. For the first claim, we are trying to prove:

$$\prod_{\mathsf{w}:\mathsf{Push}\ \mathsf{g}\ \mathsf{f}}\left\|\sum_{\mathsf{a}:\mathsf{C}}\mathsf{pushr}\ \mathsf{a}=\mathsf{w}\right\|$$

Therefore, by the induction principle for pushouts, it is enough to prove that the following types are inhabited:

$$\prod_{\mathsf{b}:\mathsf{B}}\left\|\sum_{\mathsf{a}:\mathsf{C}}\mathsf{pushr}\ \mathsf{a}=\mathsf{pushl}\ \mathsf{b}\right\|$$
$$\prod_{\mathsf{c}:\mathsf{C}}\left\|\sum_{\mathsf{a}:\mathsf{C}}\mathsf{pushr}\ \mathsf{a}=\mathsf{pushr}\ \mathsf{c}\right\|$$

The second type is inhabited by:

$$\lambda \mathsf{c} : \mathsf{C}.\ \left|(\mathsf{c}, \mathsf{refl}_{(\mathsf{Push\ g\ f})}\ (\mathsf{pushr\ c}))\right|$$

For the first type, let $\mathsf{b} : \mathsf{B}$. We know g is surjective, this means that type $\left\|\sum_{\mathsf{y}:\mathsf{A}} \mathsf{g\ y} = \mathsf{b}\right\|$ is inhabited. Therefore, we will be done if we can prove that the following type is inhabited:

$$\left\|\sum_{\mathsf{y}:\mathsf{A}} \mathsf{g\ y} = \mathsf{b}\right\| \to \left\|\sum_{\mathsf{a}:\mathsf{C}} \mathsf{pushr\ a} = \mathsf{pushl\ b}\right\|$$

By (-1)-recursion (Corollary 1.11.4) and Σ-recursion (Lemma 1.5.25), it is enough to prove:

$$\prod_{\mathsf{y}:\mathsf{A}} \left((\mathsf{g\ y} = \mathsf{b}) \to \left\|\sum_{\mathsf{a}:\mathsf{C}} \mathsf{pushr\ a} = \mathsf{pushl\ b}\right\|\right)$$

Let $\mathsf{y} : \mathsf{A}$ with $\mathsf{p}_1 : \mathsf{g\ y} = \mathsf{b}$. Therefore, $\mathsf{p}_2 :\equiv \mathsf{ap}_{\mathsf{pushl}}\ \mathsf{p}_1 : \mathsf{pushl}\ (\mathsf{g\ y}) = \mathsf{pushl\ b}$. We also have:

$$\mathsf{p}_3 :\equiv (\mathsf{pushiden\ y})^{-1} : \mathsf{pushr}\ (\mathsf{f\ y}) = \mathsf{pushl}\ (\mathsf{g\ y})$$

Hence:

$$|(\mathsf{f\ y}, \mathsf{p}_3 \centerdot \mathsf{p}_2)| : \left\|\sum_{\mathsf{a}:\mathsf{C}} \mathsf{pushr\ a} = \mathsf{pushl\ b}\right\|$$

The second claim is proved similarly.

□

1.11.5 Quotients

Let us work for a moment inside classical set theory. Given a set A and an arbitrary binary relation $R \subseteq A \times A$, we can define the quotient of A under R, denoted A/R, to be the quotient of A under the smallest equivalence relation on A containing R. This idea can be expressed in the category of sets, where set A/R is defined to be the coequalizer for:

$$R \underset{pr_2}{\overset{pr_1}{\rightrightarrows}} A$$

where pr_1 and pr_2 are the projection functions [5].

By analogy, given a type $\mathsf{A} : \mathcal{U}_i$ and a *binary relation*[66] $\mathsf{R} : \mathsf{A} \to \mathsf{A} \to \mathbf{Prop}_j$, we can define the *quotient* of A under R, denoted $\mathsf{A/R}$, to be the coequalizer for:

$$\left(\sum_{\mathsf{a}:\mathsf{A}} \sum_{\mathsf{b}:\mathsf{A}} \mathsf{R\ a\ b}\right) \underset{\mathsf{pr}_1 \circ \mathsf{pr}_2}{\overset{\mathsf{pr}_1}{\rightrightarrows}} \mathsf{A}$$

Again, it is possible to define $\mathsf{A/R}$ as a HIT by the same strategy we followed for coequalizers. Instead, $\mathsf{A/R}$ will be defined in terms of coequalizers, as was done with pushouts.

Also, on this report, type $\mathsf{A/R}$ will be required to be a set.[67] Hence, $\mathsf{A/R}$ will be defined as the 0-truncation of the coequalizer.

Lemma 1.11.24 (Formation and introduction for quotients)**.** *Let* $\mathsf{A} : \mathcal{U}_i$ *be a type and* $\mathsf{R} : \mathsf{A} \to \mathsf{A} \to \mathbf{Prop}_j$ *a binary relation.*

There is a set *at level* $\max\{i, j\}$*, denoted* $\mathsf{A/R}$*, and called the* quotient *of* A *under* R*.*

The set $\mathsf{A/R}$ *has the following two constructors:*

[66] By Lemma 1.6.29, it makes no difference if we define the type of R as $(\mathsf{A} \times \mathsf{A}) \to \mathbf{Prop}_j$ or as $\mathsf{A} \to \mathsf{A} \to \mathbf{Prop}_j$.

Also, strictly speaking, R is a binary predicate. However, in Definition 1.8.19 it was explained that it is safe to identify a predicate with the subtype or subset it defines, which in this case is $\sum_{\mathsf{a}:\mathsf{A}} \sum_{\mathsf{b}:\mathsf{A}} \mathsf{R\ a\ b}$. This Σ-type is the *real* binary relation.

[67] The reasons for this restriction will become clear in Chapter 3.

- *Term constructor.* $[\,]\colon \mathsf{A} \to \mathsf{A/R}$.
  Instead of $[\,]\ \mathsf{a}$, *we will write* $[\mathsf{a}]$. *Term* $[\mathsf{a}]$ *can be understood as the* equivalence class *of* a.
- *Higher constructor.* $\mathsf{quotiden}\colon \prod_{\mathsf{a,b:A}} (\mathsf{R\ a\ b}) \to ([\mathsf{a}] = [\mathsf{b}])$.
  Instead of $\mathsf{quotiden\ a\ b\ p}$, *we will write* $\mathsf{quotiden\ p}$, *where terms* a *and* b *are left implicit.*

Proof. Define:

$$\mathsf{A/R} :\equiv \|\mathsf{Coeq\ pr_1\ (pr_2 \circ pr_1)}\|_0$$

where:

$$\left(\sum_{\mathsf{a:A}} \sum_{\mathsf{b:A}} \mathsf{R\ a\ b}\right) \underset{\mathsf{pr_1 \circ pr_2}}{\overset{\mathsf{pr_1}}{\rightrightarrows}} \mathsf{A}$$

Notice that $\mathsf{A/R}$ is a set since it is a 0-truncation. Also, it is at level $\max\{i, j\}$ since $\mathsf{Coeq\ pr_1\ (pr_1 \circ pr_2)}$ must be at level $\max\{\max\{i, j\}, i\}$ (see Remark 1.5.29 for computing universe level on Σ-types), and 0-truncations preserve levels (see Definition 1.11.12).

Now, define:

$$[\,] :\equiv (\lambda \mathsf{a}.\ |\mathsf{coeq\ a}|_0)\colon \mathsf{A} \to \mathsf{A/R}$$

$$\mathsf{quotiden} :\equiv (\lambda \mathsf{a}.\ \lambda \mathsf{b}.\ \lambda \mathsf{p}.\ \mathsf{ap}_{|\ |_0}\ (\mathsf{coeqiden}\ (\mathsf{a}, (\mathsf{b}, \mathsf{p})))) \colon \prod_{\mathsf{a,b:A}} (\mathsf{R\ a\ b}) \to ([\mathsf{a}] = [\mathsf{b}])$$

Notice that $\mathsf{quotiden}$ is well-typed since:

$$\mathsf{coeqiden}\ (\mathsf{a}, (\mathsf{b}, \mathsf{p}))\colon \mathsf{coeq}\ (\mathsf{pr_1}\ (\mathsf{a}, (\mathsf{b}, \mathsf{p}))) = \mathsf{coeq}\ (\mathsf{pr_1}\ (\mathsf{pr_2}\ (\mathsf{a}, (\mathsf{b}, \mathsf{p}))))$$

or equivalently:

$$\mathsf{coeqiden}\ (\mathsf{a}, (\mathsf{b}, \mathsf{p}))\colon \mathsf{coeq\ a} = \mathsf{coeq\ b}$$

which means:

$$\mathsf{ap}_{|\ |_0}\ (\mathsf{coeqiden}\ (\mathsf{a}, (\mathsf{b}, \mathsf{p})))\colon |\mathsf{coeq\ a}|_0 = |\mathsf{coeq\ b}|_0$$

or equivalently:

$$\mathsf{ap}_{|\ |_0}\ (\mathsf{coeqiden}\ (\mathsf{a}, (\mathsf{b}, \mathsf{p})))\colon [\mathsf{a}] = [\mathsf{b}]$$

by definition of $[\,]$.

□

Quotients have the following (simplified) induction and recursion principles.

Lemma 1.11.25 (Induction principle for quotients)**.** *Let* $\mathsf{A}\colon \mathcal{U}_i$ *be a type,* $\mathsf{R}\colon \mathsf{A} \to \mathsf{A} \to \mathbf{Prop}_j$ *a binary relation, and* $\mathsf{P}\colon (\mathsf{A/R}) \to \mathbf{Prop}_k$ *a family of propositions.*

If the following type is inhabited:

$$\prod_{\mathsf{a:A}} \mathsf{P}\ [\mathsf{a}]$$

then the following type is inhabited:

$$\prod_{\mathsf{w:A/R}} \mathsf{P\ w}$$

Proof. We are trying to prove:

$$\prod_{\mathsf{w}:\|\mathsf{Coeq\ pr_1\ (pr_2 \circ pr_1)}\|_0} \mathsf{P\ w}$$

So, by the induction principle for 0-truncations (Definition 1.11.13), it is enough to prove:[68]

$$\prod_{\mathsf{w:Coeq\ pr_1\ (pr_2 \circ pr_1)}} \mathsf{P}\ |\mathsf{w}|_0$$

[68] Strictly speaking, the induction principle for 0-truncations requires a family $\mathsf{P'}\colon (\mathsf{A/R}) \to \mathbf{Set}_k$, which can be defined as:

$$\mathsf{P'} :\equiv \lambda \mathsf{w}.\ (\mathsf{pr_1}\ (\mathsf{P\ w}), \mathsf{m}\ (\mathsf{pr_2}\ (\mathsf{P\ w})))$$

where m is the proof for Lemma 1.8.11, which transforms a proof for "$\mathsf{pr_1}\ (\mathsf{P\ w})$ is a proposition" into a proof for "$\mathsf{pr_1}\ (\mathsf{P\ w})$ is a set". Since this is just a technicality, it is harmless to think that $\mathsf{P}\colon (\mathsf{A/R}) \to \mathbf{Prop}_k$ also has type $(\mathsf{A/R}) \to \mathbf{Set}_k$.

but, by the induction principle for coequalizers (Lemma 1.11.17), it is enough to prove:

$$\prod_{\mathsf{a}:\mathsf{A}} \mathsf{P}\ |(\mathsf{coeq}\ \mathsf{a})|_0$$

or equivalently, by definition of $[\,]$:

$$\prod_{\mathsf{a}:\mathsf{A}} \mathsf{P}\ [\mathsf{a}]$$

which is inhabited by hypothesis.

□

Lemma 1.11.26 (Recursion principle for quotients)**.** *Let* $\mathsf{A}:\mathcal{U}_i$ *be a type,* $\mathsf{R}:\mathsf{A}\to\mathsf{A}\to\mathbf{Prop}_j$ *a binary relation, and* $\mathsf{D}:\mathbf{Set}_k$ *a set.*

If there is a function:

$$\mathsf{h}:\mathsf{A}\to\mathsf{D}$$

such that the following type is inhabited:

$$\prod_{\mathsf{a},\mathsf{b}:\mathsf{A}} (\mathsf{R}\ \mathsf{a}\ \mathsf{b})\to(\mathsf{h}\ \mathsf{a}=\mathsf{h}\ \mathsf{b})$$

then there is a function:

$$\mathsf{u}:(\mathsf{A}/\mathsf{R})\to\mathsf{D}$$

satisfying (for any $\mathsf{a}:\mathsf{A}$*):*

$$\mathsf{u}\ [\mathsf{a}]\equiv\mathsf{h}\ \mathsf{a}$$

Proof. We are trying to construct a function $\mathsf{u}:\|\mathsf{Coeq}\ \mathsf{pr}_1\ (\mathsf{pr}_2\circ\mathsf{pr}_1)\|_0\to\mathsf{D}$. So, by the recursion principle for 0-truncations (Corollary 1.11.14), it is enough to construct a function $\mathsf{s}:(\mathsf{Coeq}\ \mathsf{pr}_1\ (\mathsf{pr}_2\circ\mathsf{pr}_1))\to\mathsf{D}$, so that u will be determined by:

$$\mathsf{u}\ |\mathsf{w}|_0:\equiv\mathsf{s}\ \mathsf{w}$$

for any $\mathsf{w}:\mathsf{Coeq}\ \mathsf{pr}_1\ (\mathsf{pr}_2\circ\mathsf{pr}_1)$.

To define s, by Lemma 1.11.18 it is enough to construct a function $\mathsf{t}:\mathsf{A}\to\mathsf{D}$ satisfying:

$$\prod_{\mathsf{q}:\sum_{\mathsf{a}:\mathsf{A}}\sum_{\mathsf{b}:\mathsf{A}}\mathsf{R}\ \mathsf{a}\ \mathsf{b}} \mathsf{t}\ (\mathsf{pr}_1\ \mathsf{q})=\mathsf{t}\ (\mathsf{pr}_1\ (\mathsf{pr}_2\ \mathsf{q})) \tag{1.33}$$

so that s will satisfy:

$$\mathsf{s}\ (\mathsf{coeq}\ \mathsf{a})\equiv\mathsf{t}\ \mathsf{a}$$

for any $\mathsf{a}:\mathsf{A}$.

So, define $\mathsf{t}:\equiv\mathsf{h}$. Now, we prove that t satisfies (1.33). By the induction principle for Σ-types, it is enough to prove:

$$\prod_{\mathsf{a},\mathsf{b}:\mathsf{A}}\ \prod_{\mathsf{p}:\mathsf{R}\ \mathsf{a}\ \mathsf{b}} \mathsf{t}\ (\mathsf{pr}_1\ (\mathsf{a},(\mathsf{b},\mathsf{p})))=\mathsf{t}\ (\mathsf{pr}_1\ (\mathsf{pr}_2\ (\mathsf{a},(\mathsf{b},\mathsf{p}))))$$

which simplifies to (by projections and definition of t):

$$\prod_{\mathsf{a},\mathsf{b}:\mathsf{A}} (\mathsf{R}\ \mathsf{a}\ \mathsf{b})\to(\mathsf{h}\ \mathsf{a}=\mathsf{h}\ \mathsf{b})$$

which is inhabited by hypothesis.

To finish the proof, we have by definitions:

$$\mathsf{u}\ [\mathsf{a}]\equiv\mathsf{u}\ |\mathsf{coeq}\ \mathsf{a}|_0\equiv\mathsf{s}\ (\mathsf{coeq}\ \mathsf{a})\equiv\mathsf{t}\ \mathsf{a}\equiv\mathsf{h}\ \mathsf{a}$$

□

Notice that the recursion principle expresses that any function that is compatible with the binary relation can be extended into equivalence classes.

We will use the following property regarding constructor $[\,]$.

Lemma 1.11.27. *Let* $\mathsf{A} : \mathcal{U}_i$ *be a type and* $\mathsf{R} : \mathsf{A} \to \mathsf{A} \to \mathbf{Prop}_j$ *a binary relation.*

The constructor $[\,] : \mathsf{A} \to \mathsf{A}/\mathsf{R}$ *is surjective.*

Proof. We want to prove $\prod_{\mathsf{w}:\mathsf{A}/\mathsf{R}} \| \sum_{\mathsf{a}:\mathsf{A}} [\mathsf{a}] = \mathsf{w} \|$. By the induction principle for quotients (Lemma 1.11.25), it is enough to prove:

$$\prod_{\mathsf{c}:\mathsf{A}} \left\| \sum_{\mathsf{a}:\mathsf{A}} [\mathsf{a}] = [\mathsf{c}] \right\|$$

But the following function inhabits this type:

$$\lambda \mathsf{c}.\ \left| (\mathsf{c}, \mathsf{refl}_{\mathsf{A}/\mathsf{R}}\ [\mathsf{c}]) \right|$$

□

1.12 Chapter summary

This chapter presented Homotopy Type Theory (HoTT) with the minimal amount of results required for the developments on Chapters 2 and 3.

Sections 1.1 through 1.5 introduced HoTT standard types by means of inference rules. Section 1.6 introduced the concepts of equivalence, type equivalence, the function extensionality axiom, and the univalence axiom. Section 1.7 explained the homotopy (or topological) interpretation for HoTT, and clarified the meaning of identity types. Section 1.8 introduced the concepts of contractible type, proposition, set, and the propositional resizing axiom. Sections 1.9 and 1.10 explained how inductive types and coinductive types can be encoded through initial algebras and final coalgebras, respectively. And finally, Section 1.11 introduced the concept of higher inductive type (HIT), together with some HITs that will be required on later chapters.

This chapter only scratched the surface of HoTT. The reader is invited to read the HoTT book [20] for further concepts and applications.

Chapter 2

Complete lattices and fixpoints in HoTT

This chapter formalizes in HoTT classical results regarding the existence of fixpoints for monotone functions on a complete lattice. The intention for doing such thing is twofold: first, we are going to make use of these results in Chapter 3, and second, we want to provide more evidence that HoTT is a suitable foundation for doing set-based mathematics.

Another important point this chapter emphasizes is that it is possible to do mathematics in HoTT without worrying too much about universe levels, though care must be taken in some edge cases. In other words, this chapter will highlight the "ignore universe levels" style of doing mathematics in HoTT.

This chapter is structured as follows:

- Sections 2.1 and 2.2 define the concepts of poset, monotone function and complete lattice. A couple of examples are shown as well.
- Section 2.3 provides a proof for the Knaster-Tarski Lemma, which is the main tool we will use to obtain fixpoints out of monotone functions.
- Section 2.4 constructs induction and coinduction principles as corollaries to the Knaster-Tarski Lemma.

The results on this chapter have been computer-checked in the Coq proof assistant [24]. See [26] for details on how to download the proof script file.

2.1 Posets and monotone functions

The basic concept in order theory is that of *partially ordered set*, or *poset* for short. A poset is a set together with a binary relation that is reflexive, antisymmetric, and transitive [9]. In HoTT, a poset can be formalized as follows.

Definition 2.1.1 (Poset)**.** Given a set $\mathsf{P} : \mathbf{Set}$.[1] Any term of type:[2]

$$\mathsf{PosetStr}\ \mathsf{P} :\equiv \sum_{\mathsf{R}:\mathsf{P}\to\mathsf{P}\to\mathbf{Prop}} \left[\left(\prod_{\mathsf{w}:\mathsf{P}} \mathsf{R}\ \mathsf{w}\ \mathsf{w} \right) \times \left(\prod_{\mathsf{w},\mathsf{y}:\mathsf{P}} (\mathsf{R}\ \mathsf{w}\ \mathsf{y}) \to (\mathsf{R}\ \mathsf{y}\ \mathsf{w}) \to (\mathsf{w} = \mathsf{y}) \right) \times \left(\prod_{\mathsf{w},\mathsf{y},\mathsf{z}:\mathsf{P}} (\mathsf{R}\ \mathsf{w}\ \mathsf{y}) \to (\mathsf{R}\ \mathsf{y}\ \mathsf{z}) \to (\mathsf{R}\ \mathsf{w}\ \mathsf{z}) \right) \right]$$

will be called a *poset structure* for P. In particular, a *poset* will be any term of type:

$$\mathsf{Poset} :\equiv \sum_{\mathsf{Q}:\mathbf{Set}} \mathsf{PosetStr}\ \mathsf{Q}$$

▲

[1] Notice we are ignoring the universe level of **Set**.

[2] By Lemma 1.6.29, it makes no difference if we define R as having type $\mathsf{P} \to \mathsf{P} \to \mathbf{Prop}$ or type $\mathsf{P} \times \mathsf{P} \to \mathbf{Prop}$.
Also, notice we are using Conventions 1.8.5 and 1.8.10 regarding implicit casting of propositions and sets. See also Definitions 1.8.4 and 1.8.9.

In other words, a poset structure over a set P will be a binary relation[3] $\mathsf{R} : \mathsf{P} \to \mathsf{P} \to \mathbf{Prop}$ together with *proofs* that R is reflexive, antisymmetric, and transitive. A poset will be a set together with a poset structure over it.

Convention 2.1.2. Given a poset $(\mathsf{P}, (\mathsf{R}, (\mathsf{r}, (\mathsf{a}, \mathsf{t})))) : \sum_{\mathsf{Q}:\mathbf{Set}} \mathsf{PosetStr}\ \mathsf{Q}$, we will write this poset as $(\mathsf{P}, \mathsf{R}, \mathsf{r}, \mathsf{a}, \mathsf{t})$ to improve readability. Whenever we want to leave terms $\mathsf{r}, \mathsf{a}, \mathsf{t}$ implicit, we will write the poset as $(\mathsf{P}, \mathsf{R}, \ldots)$.

Also, relation R can be written using the symbol $\leq$ or some other binary relation symbol. On those cases, we will use infix notation, i.e. instead of $(\leq\ \mathsf{w}\ \mathsf{y})$, we will write $(\mathsf{w} \leq \mathsf{y})$.

▲

In the definition of $\mathsf{PosetStr}\ \mathsf{P}$, the identity type $\mathsf{w} = \mathsf{y}$ is a mere proposition for any $\mathsf{w}, \mathsf{y} : \mathsf{P}$, because P is a set (see Lemma 1.8.17). Also, since $\mathsf{R}\ \mathsf{w}\ \mathsf{y}$ is a mere proposition for any $\mathsf{w}, \mathsf{y} : \mathsf{P}$, by closure properties for propositions (Lemma 1.8.17) the three types in $\mathsf{PosetStr}\ \mathsf{P}$ expressing that R is reflexive, antisymmetric, and transitive, are all mere propositions. This means that any proof for the types expressing that R is reflexive, antisymmetric, and transitive, will be as good as any other one. This is in accordance with classical reasoning in set-based mathematics, where we do not care about all the different ways in which the binary relation satisfies the properties, as long as the relation satisfies them (i.e. classical mathematics is *proof-irrelevant*). Therefore, HoTT is able to capture this aspect of classical mathematics.

To emphasize the previous point, let us suppose $\mathsf{P} : \mathcal{U}$ is a type (not necessarily a set), and let us change the binary relation $\mathsf{R} : \mathsf{P} \to \mathsf{P} \to \mathbf{Prop}$ in the definition of $\mathsf{PosetStr}\ \mathsf{P}$ to a type family $\mathsf{R} : \mathsf{P} \to \mathsf{P} \to \mathcal{U}$. Then, the three types expressing that R is reflexive, antisymmetric, and transitive, are *not* necessarily mere propositions. Let us explore the consequences of this change.

Suppose we construct a type family $\mathsf{S} : \mathsf{P} \to \mathsf{P} \to \mathcal{U}$, together with two "poset structures" $(\mathsf{S}, \mathsf{r}_1, \mathsf{a}_1, \mathsf{t}_1) : \mathsf{PosetStr}\ \mathsf{P}$ and $(\mathsf{S}, \mathsf{r}_2, \mathsf{a}_2, \mathsf{t}_2) : \mathsf{PosetStr}\ \mathsf{P}$. These two poset structures will *not* be necessarily the same. For example, it might *not* be the case that $\mathsf{r}_1 = \mathsf{r}_2$, because type $\prod_{\mathsf{w}:\mathsf{P}} \mathsf{S}\ \mathsf{w}\ \mathsf{w}$ is not necessarily a mere proposition. This situation is completely at odds with classical mathematics, because we have a relation S that is reflexive, antisymmetric and transitive, but we are claiming that S *itself* is somehow two different poset structures!

The key to understand this "inconsistency" is to notice that when the statement "S is reflexive, antisymmetric and transitive" is claimed in standard mathematics, we are implicitly accepting that any proof for this claim will be as good as any other one. This classical assumption of "proof irrelevancy" is incompatible with our example, since we have two non-identifiable proofs (r_1 and r_2) that S is reflexive.

This means that our attempt of generalizing the definition of poset structure from binary relations into general type families, is not capturing the classical concept of poset structure, because in classical mathematics we *require* poset structures $(\mathsf{S}, \mathsf{r}_1, \mathsf{a}_1, \mathsf{t}_1)$ and $(\mathsf{S}, \mathsf{r}_2, \mathsf{a}_2, \mathsf{t}_2)$ to be the same.

HoTT can express properties about proofs themselves. In our example, proofs r_1 and r_2 are different, a possibility that is not even considered in classical mathematics. In HoTT we do *proof-relevant* mathematics, and when we want to represent the proof-irrelevant behavior of classical mathematics, we work in the universe of sets (**Set**) and propositions (**Prop**).

To convince us that Definition 2.1.1 does capture the required proof-irrelevance behavior of classical posets, just notice that Lemma 1.8.14 implies that any two poset structures $(\mathsf{S}, \mathsf{r}_1, \mathsf{a}_1, \mathsf{t}_1)$ and $(\mathsf{S}, \mathsf{r}_2, \mathsf{a}_2, \mathsf{t}_2)$ are identifiable, because the three types expressing reflexivity, antisymmetry and transitivity are mere propositions, and we have a trivial proof for $\mathsf{S} = \mathsf{S}$.

There is another point regarding the definition of $\mathsf{PosetStr}\ \mathsf{P}$. Since **Prop** is a set (Lemma 1.8.18), by closure properties for Π-types and Σ-types (Lemma 1.8.18), $\mathsf{PosetStr}\ \mathsf{P}$ is a set for any set P. Therefore, type $\mathsf{PosetStr}\ \mathsf{P}$ can also be interpreted as "the *set* of all poset structures over set P".

Now, we present an example of poset.

Example 2.1.3 (Dual Poset)**.** Given a poset $(\mathsf{P}, \leq, \mathsf{rh}, \mathsf{ah}, \mathsf{th})$, its dual poset will also have P as underlying set, but its relation will be inverted, i.e. $\mathsf{w} \leq \mathsf{y}$ will hold in the dual if and only if $\mathsf{y} \leq \mathsf{w}$ holds in P.

Therefore, we define the dual binary relation as follows:

$$\leq^{-1} :\equiv (\lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \mathsf{y} \leq \mathsf{w}) : \mathsf{P} \to \mathsf{P} \to \mathbf{Prop}$$

Now, we need to prove that there is some term inhabiting each one of the types expressing that $\leq^{-1}$ is reflexive, antisymmetric, and transitive.

[3] Strictly speaking, R is a binary predicate. However, in Definition 1.8.19 it was explained that it is safe to identify a predicate with the subtype or subset it defines, which in this case is $\sum_{\mathsf{a}:\mathsf{P}} \sum_{\mathsf{b}:\mathsf{P}} \mathsf{R}\ \mathsf{a}\ \mathsf{b}$. This Σ-type is the *real* binary relation.

- $\leq^{-1}$ is reflexive, i.e. we need to prove that type $\prod_{\mathsf{w}:\mathsf{P}} \mathsf{w} \leq^{-1} \mathsf{w}$ is inhabited.
  Given $\mathsf{w}:\mathsf{P}$, we have by definition $(\mathsf{w} \leq^{-1} \mathsf{w}) \equiv (\mathsf{w} \leq \mathsf{w})$. But $\leq$ is reflexive, i.e. we have a proof:
  $$\mathsf{rh}:\prod_{\mathsf{a}:\mathsf{P}} \mathsf{a} \leq \mathsf{a}$$
  Therefore, $\mathsf{rh}\ \mathsf{w}:\mathsf{w} \leq \mathsf{w}$, which means that $\mathsf{w} \leq^{-1} \mathsf{w}$ is inhabited.
- $\leq^{-1}$ is antisymmetric, i.e. we need to prove that the following type is inhabited:
  $$\prod_{\mathsf{w},\mathsf{y}:\mathsf{P}} (\mathsf{w} \leq^{-1} \mathsf{y}) \to (\mathsf{y} \leq^{-1} \mathsf{w}) \to (\mathsf{w} = \mathsf{y})$$
  Given $\mathsf{w},\mathsf{y}:\mathsf{P}$ with $\mathsf{h}_1:\mathsf{w} \leq^{-1} \mathsf{y}$ and $\mathsf{h}_2:\mathsf{y} \leq^{-1} \mathsf{w}$, we have by definition:
  $$(\mathsf{w} \leq^{-1} \mathsf{y}) \equiv (\mathsf{y} \leq \mathsf{w})$$
  $$(\mathsf{y} \leq^{-1} \mathsf{w}) \equiv (\mathsf{w} \leq \mathsf{y})$$
  But $\leq$ is antisymmetric, i.e. we have a proof:
  $$\mathsf{ah}:\prod_{\mathsf{a},\mathsf{b}:\mathsf{P}} (\mathsf{a} \leq \mathsf{b}) \to (\mathsf{b} \leq \mathsf{a}) \to (\mathsf{a} = \mathsf{b})$$
  Therefore, $\mathsf{ah}\ \mathsf{w}\ \mathsf{y}\ \mathsf{h}_2\ \mathsf{h}_1:\mathsf{w} = \mathsf{y}$.
- $\leq^{-1}$ is transitive, i.e. we need to prove that the following type is inhabited:
  $$\prod_{\mathsf{w},\mathsf{y},\mathsf{z}:\mathsf{P}} (\mathsf{w} \leq^{-1} \mathsf{y}) \to (\mathsf{y} \leq^{-1} \mathsf{z}) \to (\mathsf{w} \leq^{-1} \mathsf{z})$$
  Given $\mathsf{w},\mathsf{y},\mathsf{z}:\mathsf{P}$ with $\mathsf{h}_1:\mathsf{w} \leq^{-1} \mathsf{y}$ and $\mathsf{h}_2:\mathsf{y} \leq^{-1} \mathsf{z}$, we have by definition:
  $$(\mathsf{w} \leq^{-1} \mathsf{y}) \equiv (\mathsf{y} \leq \mathsf{w})$$
  $$(\mathsf{y} \leq^{-1} \mathsf{z}) \equiv (\mathsf{z} \leq \mathsf{y})$$
  $$(\mathsf{w} \leq^{-1} \mathsf{z}) \equiv (\mathsf{z} \leq \mathsf{w})$$
  But R is transitive, i.e. we have a proof:
  $$\mathsf{th}:\prod_{\mathsf{a},\mathsf{b},\mathsf{c}:\mathsf{P}} (\mathsf{a} \leq \mathsf{b}) \to (\mathsf{b} \leq \mathsf{c}) \to (\mathsf{a} \leq \mathsf{c})$$
  Therefore, $\mathsf{th}\ \mathsf{z}\ \mathsf{y}\ \mathsf{w}\ \mathsf{h}_2\ \mathsf{h}_1:\mathsf{z} \leq \mathsf{w}$, or equivalently, $\mathsf{w} \leq^{-1} \mathsf{z}$ is inhabited.

▲

The next example shows that any subset of a poset is still a poset under the same order relation. This example will show how we work in HoTT with the concept of *subset*. As was discussed in Definition 1.8.19, subsets of a set P can be identified with terms of type $\mathsf{P} \to \mathbf{Prop}$ (i.e. predicates or properties on P).

Example 2.1.4 (Subset Poset)**.** Let $(\mathsf{P}, \leq, \mathsf{rh}, \mathsf{ah}, \mathsf{th})$ be a poset, and $\mathsf{S}:\mathsf{P} \to \mathbf{Prop}$ a subset (predicate) of P. Then we can form a poset $(\mathsf{Q}, \leq_{\mathsf{Q}}, \ldots)$, where Q is the subset defined by predicate S, i.e. $\mathsf{Q} :\equiv \sum_{\mathsf{w}:\mathsf{P}} \mathsf{S}\ \mathsf{w}$, and $\leq_{\mathsf{Q}}$ is relation $\leq$ restricted to subset S, i.e.,

$$\leq_{\mathsf{Q}} :\equiv (\lambda \mathsf{w}.\ \lambda \mathsf{y}.\ \mathsf{pr}_1\ \mathsf{w} \leq \mathsf{pr}_1\ \mathsf{y}):\mathsf{Q} \to \mathsf{Q} \to \mathbf{Prop}$$

Now, we need to show that $\leq_{\mathsf{Q}}$ is an order relation.

- $\leq_{\mathsf{Q}}$ is reflexive, i.e. we need to show that type $\prod_{\mathsf{w}:\mathsf{Q}} \mathsf{w} \leq_{\mathsf{Q}} \mathsf{w}$ is inhabited.
  Let $\mathsf{w}:\mathsf{Q}$. By definition, $(\mathsf{w} \leq_{\mathsf{Q}} \mathsf{w}) \equiv (\mathsf{pr}_1\ \mathsf{w} \leq \mathsf{pr}_1\ \mathsf{w})$. But R is reflexive, i.e. we have a proof $\mathsf{rh}:\prod_{\mathsf{a}:\mathsf{P}} \mathsf{a} \leq \mathsf{a}$. Therefore, $\mathsf{rh}\ (\mathsf{pr}_1\ \mathsf{w}):\mathsf{pr}_1\ \mathsf{w} \leq \mathsf{pr}_1\ \mathsf{w}$, or equivalently, $\mathsf{w} \leq_{\mathsf{Q}} \mathsf{w}$ is inhabited.
- $\leq_{\mathsf{Q}}$ is antisymmetric, i.e. we need to show that type $\prod_{\mathsf{w},\mathsf{y}:\mathsf{Q}} (\mathsf{w} \leq_{\mathsf{Q}} \mathsf{y}) \to (\mathsf{y} \leq_{\mathsf{Q}} \mathsf{w}) \to (\mathsf{w} = \mathsf{y})$ is inhabited.
  Let $\mathsf{w},\mathsf{y}:\mathsf{Q}$ with $\mathsf{h}_1:\mathsf{w} \leq_{\mathsf{Q}} \mathsf{y}$ and $\mathsf{h}_2:\mathsf{y} \leq_{\mathsf{Q}} \mathsf{w}$. By definition:
  $$(\mathsf{w} \leq_{\mathsf{Q}} \mathsf{y}) \equiv (\mathsf{pr}_1\ \mathsf{w} \leq \mathsf{pr}_1\ \mathsf{y})$$
  $$(\mathsf{y} \leq_{\mathsf{Q}} \mathsf{w}) \equiv (\mathsf{pr}_1\ \mathsf{y} \leq \mathsf{pr}_1\ \mathsf{w})$$
  But R is antisymmetric, i.e. we have a proof:
  $$\mathsf{ah}:\prod_{\mathsf{a},\mathsf{b}:\mathsf{P}} (\mathsf{a} \leq \mathsf{b}) \to (\mathsf{b} \leq \mathsf{a}) \to (\mathsf{a} = \mathsf{b})$$
  Therefore, $\mathsf{ah}\ (\mathsf{pr}_1\ \mathsf{w})\ (\mathsf{pr}_1\ \mathsf{y})\ \mathsf{h}_1\ \mathsf{h}_2:\mathsf{pr}_1\ \mathsf{w} = \mathsf{pr}_1\ \mathsf{y}$.
  Now, by Lemma 1.8.14, type $\mathsf{w} = \mathsf{y}$ is inhabited, because $\mathsf{S}\ \mathsf{z}$ is a mere proposition for every $\mathsf{z}:\mathsf{P}$ and $\mathsf{pr}_1\ \mathsf{w} = \mathsf{pr}_1\ \mathsf{y}$ is inhabited.
- $\leq_{\mathsf{Q}}$ is transitive, i.e. we need to show that type $\prod_{\mathsf{w},\mathsf{y},\mathsf{z}:\mathsf{Q}} (\mathsf{w} \leq_{\mathsf{Q}} \mathsf{y}) \to (\mathsf{y} \leq_{\mathsf{Q}} \mathsf{z}) \to (\mathsf{w} \leq_{\mathsf{Q}} \mathsf{z})$ is inhabited.
  Let $\mathsf{w},\mathsf{y},\mathsf{z}:\mathsf{Q}$ with $\mathsf{h}_1:\mathsf{w} \leq_{\mathsf{Q}} \mathsf{y}$ and $\mathsf{h}_2:\mathsf{y} \leq_{\mathsf{Q}} \mathsf{z}$. By definition:
  $$(\mathsf{w} \leq_{\mathsf{Q}} \mathsf{y}) \equiv (\mathsf{pr}_1\ \mathsf{w} \leq \mathsf{pr}_1\ \mathsf{y})$$
  $$(\mathsf{y} \leq_{\mathsf{Q}} \mathsf{z}) \equiv (\mathsf{pr}_1\ \mathsf{y} \leq \mathsf{pr}_1\ \mathsf{z})$$
  $$(\mathsf{w} \leq_{\mathsf{Q}} \mathsf{z}) \equiv (\mathsf{pr}_1\ \mathsf{w} \leq \mathsf{pr}_1\ \mathsf{z})$$
  But R is transitive, i.e. we have a proof:
  $$\mathsf{th}:\prod_{\mathsf{a},\mathsf{b},\mathsf{c}:\mathsf{P}} (\mathsf{a} \leq \mathsf{b}) \to (\mathsf{b} \leq \mathsf{c}) \to (\mathsf{a} \leq \mathsf{c})$$
  Therefore, $\mathsf{th}\ (\mathsf{pr}_1\ \mathsf{w})\ (\mathsf{pr}_1\ \mathsf{y})\ (\mathsf{pr}_1\ \mathsf{z})\ \mathsf{h}_1\ \mathsf{h}_2:\mathsf{pr}_1\ \mathsf{w} \leq \mathsf{pr}_1\ \mathsf{z}$, or equivalently, $\mathsf{w} \leq_{\mathsf{Q}} \mathsf{z}$ is inhabited.

▲

Another important example from order theory is the powerset poset, which consists on all subsets of a set ordered by inclusion. In HoTT this is achieved by constructing a poset on type $\mathsf{S} \to \mathbf{Prop}$ (the type of all subsets or predicates on set S). Since **Prop** is a set (Lemma 1.8.18), by closure properties for the arrow type (Lemma 1.8.18), $\mathsf{T} \to \mathbf{Prop}$ will be a set for *any type* $\mathsf{T}:\mathcal{U}$. This means we can form not only the powerset poset for any set $\mathsf{S}:\mathbf{Set}$, but also the "powertype" poset for any *type* $\mathsf{T}:\mathcal{U}$, as the following example shows.

Example 2.1.5 ("Powertype" Poset)**.** Let $\mathsf{T}:\mathcal{U}$ be a type. Then, we can build a poset $(\mathsf{P},\subseteq,\ldots)$, where P is the *set* of all subtypes (predicates) of T, i.e. $\mathsf{P} :\equiv \mathsf{T} \to \mathbf{Prop}$, and $\subseteq$ is the subtype "inclusion" relation:

$$\subseteq :\equiv \left(\lambda \mathsf{A}.\ \lambda \mathsf{B}.\ \prod_{\mathsf{w}:\mathsf{T}} (\mathsf{A}\ \mathsf{w}) \to (\mathsf{B}\ \mathsf{w})\right) : \mathsf{P} \to \mathsf{P} \to \mathbf{Prop}$$

Given two predicates $\mathsf{A},\mathsf{B}:\mathsf{P}$, the type $\prod_{\mathsf{w}:\mathsf{T}} \mathsf{A}\ \mathsf{w} \to \mathsf{B}\ \mathsf{w}$ is indeed a mere proposition, since $\mathsf{B}\ \mathsf{w}$ is a mere proposition for any $\mathsf{w}:\mathsf{T}$, and mere propositions are closed under Π-types (Lemma 1.8.17). Therefore, $\subseteq$ is well-typed.

Also, since $\prod_{\mathsf{w}:\mathsf{T}} (\mathsf{A}\ \mathsf{w}) \to (\mathsf{B}\ \mathsf{w})$ is a mere proposition for any $\mathsf{A},\mathsf{B}:\mathsf{P}$, it can be interpreted as a logical statement. $\mathsf{A} \subseteq \mathsf{B}$ reads as follows: "For any $\mathsf{w}:\mathsf{T}$, if w is in A (or w satisfies A) then w is in B (or w satisfies B)". Therefore, $\mathsf{A} \subseteq \mathsf{B}$ means that A is "contained" in B.

We proceed to show that $\subseteq$ is an order relation.

- $\subseteq$ is reflexive, i.e. we need to show that type $\prod_{A:P} A \subseteq A$ is inhabited.
  Let $A : P$. By definition, $(A \subseteq A) \equiv \prod_{w:T} (A\ w) \to (A\ w)$.
  So, let $w : T$ such that $A\ w$ holds. But then, $A\ w$ holds trivially by hypothesis.
- $\subseteq$ is antisymmetric, i.e. we need to show that type $\prod_{A,B:P} (A \subseteq B) \to (B \subseteq A) \to (A = B)$ is inhabited.
  Let $A, B : P$ with $h_1 : A \subseteq B$ and $h_2 : B \subseteq A$. By definition:

$$(A \subseteq B) \equiv \prod_{w:T} (A\ w) \to (B\ w)$$
$$(B \subseteq A) \equiv \prod_{w:T} (B\ w) \to (A\ w)$$

  Claim 1: Type $\prod_{w:T} A\ w = B\ w$ is inhabited.
  Let $w : T$. We know $A\ w$ and $B\ w$ are mere propositions. Also, $(A\ w) \to (B\ w)$ and $(B\ w) \to (A\ w)$ hold by hypothesis. Therefore, by Lemma 1.8.8, $A\ w$ and $B\ w$ are equivalent types. Hence, by univalence (see Section 1.6.2), we get $A\ w = B\ w$.
  This proves the claim.

  To finish the proof, since A and B are functions (i.e. $A, B : T \to \mathbf{Prop}$), by Claim 1 and function extensionality (see Section 1.6.1), we get $A = B$.
- $\subseteq$ is transitive, i.e. we need to show that type $\prod_{A,B,C:P} (A \subseteq B) \to (B \subseteq C) \to (A \subseteq C)$ is inhabited.
  Let $A, B, C : P$ with $h_1 : A \subseteq B$ and $h_2 : B \subseteq C$. By definition:

$$(A \subseteq B) \equiv \prod_{w:T} (A\ w) \to (B\ w)$$
$$(B \subseteq C) \equiv \prod_{w:T} (B\ w) \to (C\ w)$$
$$(A \subseteq C) \equiv \prod_{w:T} (A\ w) \to (C\ w)$$

  So, let $w : T$ with $h_3 : A\ w$. This implies, $h_1\ w : (A\ w) \to (B\ w)$ and $h_2\ w : (B\ w) \to (C\ w)$.
  Therefore, $h_2\ w\ (h_1\ w\ h_3) : C\ w$ as was required.

▲

The other basic concept in order theory is that of *monotone function* between two posets (also called *order-preserving functions*).

Definition 2.1.6 (Monotone function)**.** Let $(P, \leq_P, r_1, a_1, t_1)$ and $(Q, \leq_Q, r_2, a_2, t_2)$ be posets, and $f : P \to Q$ a function. If the following type is inhabited:

$$\prod_{w,y:P} (w \leq_P y) \to (f\ w \leq_Q f\ y)$$

then f is called *monotone*.

▲

Again, since $a \leq_Q b$ is a mere proposition for any $a, b : Q$, by closure properties for mere propositions (Lemma 1.8.17), type $\prod_{w,y:P} (w \leq_P y) \to (f\ w \leq_Q f\ y)$ is a mere proposition. Therefore, it reads as: "If w is less than y, then $(f\ w)$ is less than $(f\ y)$" (i.e. f preserves the order between w and y).

Convention 2.1.7. From this moment on, unless otherwise stated, all monotone functions will have the *same* poset as domain and range.

▲

Monotone functions are still monotone if we replace their poset by their dual poset (Example 2.1.3), as the following lemma shows.

Lemma 2.1.8. *Let* $\mathsf{A} :\equiv (\mathsf{P}, \leq, \mathsf{r}_1, \mathsf{a}_1, \mathsf{t}_1)$ *be a poset, and* $\mathsf{f} : \mathsf{P} \to \mathsf{P}$ *a monotone function. Let* $\mathsf{A}^{-1} :\equiv (\mathsf{P}, \leq^{-1}, \mathsf{r}_2, \mathsf{a}_2, \mathsf{t}_2)$ *be the dual poset, as constructed in Example 2.1.3. Then, f is monotone in the dual poset, i.e. the following type is inhabited:*

$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{P}} (\mathsf{w} \leq^{-1} \mathsf{y}) \to (\mathsf{f}\ \mathsf{w} \leq^{-1} \mathsf{f}\ \mathsf{y})$$

Proof. Let $\mathsf{w}, \mathsf{y} : \mathsf{P}$ such that $\mathsf{h}_1 : \mathsf{w} \leq^{-1} \mathsf{y}$. By definition, $(\mathsf{w} \leq^{-1} \mathsf{y}) \equiv (\mathsf{y} \leq \mathsf{w})$ and $(\mathsf{f}\ \mathsf{w} \leq^{-1} \mathsf{f}\ \mathsf{y}) \equiv (\mathsf{f}\ \mathsf{y} \leq \mathsf{f}\ \mathsf{w})$.

But f is monotone in A, i.e. we have a proof $\mathsf{k} : \prod_{\mathsf{a},\mathsf{b}:\mathsf{P}} (\mathsf{a} \leq \mathsf{b}) \to (\mathsf{f}\ \mathsf{a} \leq \mathsf{f}\ \mathsf{b})$. So, we have $\mathsf{k}\ \mathsf{y}\ \mathsf{w}\ \mathsf{h}_1 : \mathsf{f}\ \mathsf{y} \leq \mathsf{f}\ \mathsf{w}$, which means $(\mathsf{f}\ \mathsf{w} \leq^{-1} \mathsf{f}\ \mathsf{y})$ is inhabited.

□

2.2 Complete lattices

A *complete lattice* is a poset where every subset has a least upper bound and a greatest lower bound [9]. It is possible to work with a more compact definition for complete lattices, because it is only required the existence of least upper bounds (see [9]). In spite of this, the author took the decision of working with the full definition, so that all constructions are more explicit and symmetrical.

To define a complete lattice, we need to define upper and lower bounds first.

Definition 2.2.1 (Upper bound and lower bound)**.** Let $(\mathsf{P}, \leq, \mathsf{r}, \mathsf{a}, \mathsf{t})$ be a poset, $\mathsf{w} : \mathsf{P}$ an element in the poset, and $\mathsf{S} : \mathsf{P} \to \mathbf{Prop}$ a subset (predicate) of P.

(i) If the following type is inhabited:

$$\mathsf{IsUpperBound}\ \mathsf{w}\ \mathsf{S} :\equiv \prod_{\mathsf{y}:\mathsf{P}} (\mathsf{S}\ \mathsf{y}) \to (\mathsf{y} \leq \mathsf{w})$$

we say that w is an *upper bound* for subset S.

(ii) If the following type is inhabited:

$$\mathsf{IsLowerBound}\ \mathsf{w}\ \mathsf{S} :\equiv \prod_{\mathsf{y}:\mathsf{P}} (\mathsf{S}\ \mathsf{y}) \to (\mathsf{w} \leq \mathsf{y})$$

we say that w is a *lower bound* for subset S.

▲

Convention 2.2.2. In the notation (IsUpperBound w S) and (IsLowerBound w S) above, the order relation is not specified. There will be no confusion on this, since the order relation will be clear from context most of the time. Nevertheless, there will be proofs where two or more posets are used at the same time. On those cases, we will emphasize the order relation $\leq$ by writing $(\mathsf{IsUpperBound}^{\leq}\ \mathsf{w}\ \mathsf{S})$ and $(\mathsf{IsLowerBound}^{\leq}\ \mathsf{w}\ \mathsf{S})$ in superscript.

▲

Again, types (IsUpperBound w S) and (IsLowerBound w S) are mere propositions, since $\mathsf{w} \leq \mathsf{y}$ is a mere proposition for any $\mathsf{w}, \mathsf{y} : \mathsf{P}$, and mere propositions are closed under Π-types (Lemma 1.8.17). Therefore, both types can be interpreted as logical statements. The first type reads: "Any $\mathsf{y} : \mathsf{P}$ that is in subset S is below w". The second type reads: "Any $\mathsf{y} : \mathsf{P}$ that is in subset S is above w".

Definition 2.2.3 (Supremum and infimum)**.** Let $(\mathsf{P}, \leq, \mathsf{r}, \mathsf{a}, \mathsf{t})$ be a poset, $\mathsf{w} : \mathsf{P}$ an element in the poset, and $\mathsf{S} : \mathsf{P} \to \mathbf{Prop}$ a subset (predicate) of P.

(i) If the following type is inhabited:

$$\mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S} :\equiv (\mathsf{IsUpperBound}\ \mathsf{w}\ \mathsf{S}) \times \left(\prod_{\mathsf{y}:\mathsf{P}} (\mathsf{IsUpperBound}\ \mathsf{y}\ \mathsf{S}) \to (\mathsf{w} \leq \mathsf{y}) \right)$$

we say that w is the *least upper bound* for subset S. Term w is also called the *supremum* of S.

(ii) If the following type is inhabited:

$$\mathsf{IsGLB}\ \mathsf{w}\ \mathsf{S} :\equiv (\mathsf{IsLowerBound}\ \mathsf{w}\ \mathsf{S}) \times \left(\prod_{\mathsf{y}:\mathsf{P}} (\mathsf{IsLowerBound}\ \mathsf{y}\ \mathsf{S}) \to (\mathsf{y} \leq \mathsf{w})\right)$$

we say that w is the *greatest lower bound* for subset S. Term w is also called the *infimum* of S.

▲

Convention 2.2.4. The same applies as in Convention 2.2.2. When the order relation $\leq$ is not clear from context, we will emphasize it as $(\mathsf{IsLUB}^{\leq}\ \mathsf{w}\ \mathsf{S})$ and $(\mathsf{IsGLB}^{\leq}\ \mathsf{w}\ \mathsf{S})$, written in superscript.

▲

Again, both components in types $(\mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S})$ and $(\mathsf{IsGLB}\ \mathsf{w}\ \mathsf{S})$ are mere propositions. $(\mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S})$ reads: "w is an upper bound for subset S and any other upper bound for S is above w". $(\mathsf{IsGLB}\ \mathsf{w}\ \mathsf{S})$ reads: "w is a lower bound for subset S and any other lower bound for S is below w".

Also, notice we used the definite article in the definitions of $(\mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S})$ and $(\mathsf{IsGLB}\ \mathsf{w}\ \mathsf{S})$ (i.e. we said *the* least upper bound and *the* greatest lower bound). This can be justified as follows. Let a and b be two least upper bounds for subset S. Then, since both of them are upper bounds and least upper bounds, both $\mathsf{a} \leq \mathsf{b}$ and $\mathsf{b} \leq \mathsf{a}$ will hold. Therefore, by antisymmetry, $\mathsf{a} = \mathsf{b}$ will be inhabited. Hence, the least upper bound is unique (the argument is similar for the greatest lower bound).

Now, the concept of complete lattice can be stated.

Definition 2.2.5 (Complete lattice)**.** Let $\mathsf{A} :\equiv (\mathsf{L}, \leq, \mathsf{r}, \mathsf{a}, \mathsf{t})$ be a poset. Any term of type:

$$\mathsf{CLatticeStr}\ \mathsf{A} :\equiv \left(\sum_{\mathsf{sup}:(\mathsf{L}\to\mathbf{Prop})\to\mathsf{L}} \prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \mathsf{IsLUB}\ (\mathsf{sup}\ \mathsf{S})\ \mathsf{S}\right) \times \left(\sum_{\mathsf{inf}:(\mathsf{L}\to\mathbf{Prop})\to\mathsf{L}} \prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \mathsf{IsGLB}\ (\mathsf{inf}\ \mathsf{S})\ \mathsf{S}\right)$$

will be called a *complete lattice structure* for poset A. In particular, any term of type:

$$\mathsf{CLattice} :\equiv \sum_{\mathsf{B}:\mathbf{Poset}} \mathsf{CLatticeStr}\ \mathsf{B}$$

will be called a *complete lattice*.

▲

In other words, to build a complete lattice structure on a poset $\mathsf{A} :\equiv (\mathsf{L}, \leq, \mathsf{r}, \mathsf{a}, \mathsf{t})$, we need to provide two functions $\mathsf{sup}, \mathsf{inf} : (\mathsf{L} \to \mathbf{Prop}) \to \mathsf{L}$, which assign to *each* subset S of L the elements $\mathsf{sup}\ \mathsf{S} : \mathsf{L}$ and $\mathsf{inf}\ \mathsf{S} : \mathsf{L}$, in such a way that $\mathsf{sup}\ \mathsf{S}$ is the supremum of S and $\mathsf{inf}\ \mathsf{S}$ is the infimum of S. A complete lattice is a poset with a complete lattice structure.

Convention 2.2.6. Given a complete lattice A, we will write its components as $(\mathsf{L}, \leq, \mathsf{r}, \mathsf{a}, \mathsf{t}; \mathsf{sup}, \mathsf{sp}, \mathsf{inf}, \mathsf{ip})$ instead of $((\mathsf{L}, \leq, \mathsf{r}, \mathsf{a}, \mathsf{t}), ((\mathsf{sup}, \mathsf{sp}), (\mathsf{inf}, \mathsf{ip})))$, so that the expression becomes easier to read. In the simplified notation, the semicolon symbol (;) separates the poset components from the complete lattice components.

We will write $(\mathsf{L}, \leq, \mathsf{r}, \mathsf{a}, \mathsf{t}; \ldots)$, whenever terms $\mathsf{sup}, \mathsf{sp}, \mathsf{inf}, \mathsf{ip}$ are left implicit.

Also, observe that A is a poset by removing its complete lattice components, i.e. by keeping only $(\mathsf{L}, \leq, \mathsf{r}, \mathsf{a}, \mathsf{t})$.

▲

To a reader familiar with the classical notion of complete lattice, the way $\mathsf{CLatticeStr}\ \mathsf{A}$ was defined may seem a bit odd. In standard mathematics, a complete lattice is defined in terms of existentials, i.e. a poset where every subset *has* a supremum and an infimum. Why are we using functions $\mathsf{sup}, \mathsf{inf} : (\mathsf{L} \to \mathbf{Prop}) \to \mathsf{L}$ in the definition of $\mathsf{CLatticeStr}\ \mathsf{A}$? If we want $\mathsf{CLatticeStr}\ \mathsf{A}$ to be closer to the classical definition (i.e. by using existentials), it should be defined as follows:

$$\mathsf{CLatticeStr}_2\ \mathsf{A} :\equiv \left(\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}\right) \times \left(\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsGLB}\ \mathsf{w}\ \mathsf{S}\right)$$

which reads: "Every subset $\mathsf{S} : \mathsf{L} \to \mathbf{Prop}$ has some $\mathsf{w} : \mathsf{L}$ acting as supremum for S, and some $\mathsf{w} : \mathsf{L}$ acting as infimum for S".

It turns out that $\mathsf{CLatticeStr}\ \mathsf{A}$ and $\mathsf{CLatticeStr}_2\ \mathsf{A}$ are equivalent types. Just apply Theorem 1.11.6 (the "Choice Theorem") to both components of $\mathsf{CLatticeStr}_2\ \mathsf{A}$ to get the required equivalence.[4] So, answering the question posed above, definition $\mathsf{CLatticeStr}\ \mathsf{A}$ was chosen because of technical convenience, i.e. we have two functions directly accessing the supremum and infimum of a subset, instead of indirectly using projection functions if we were to use definition $\mathsf{CLatticeStr}_2\ \mathsf{A}$.

The reader may object that $\mathsf{CLatticeStr}_2\ \mathsf{A}$ is not "classical enough", since all its Σ-types are constructive existentials (see Remark 1.5.30 for an explanation). To make $\mathsf{CLatticeStr}_2\ \mathsf{A}$ truly classical, we have to (-1)-truncate all sigmas, so that they become classical existentials (see Remark 1.11.5). Therefore, we have a third candidate for the definition of complete lattice structure:

$$\mathsf{CLatticeStr}_3\ \mathsf{A} :\equiv \left(\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \left\|\sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}\right\|\right) \times \left(\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \left\|\sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsGLB}\ \mathsf{w}\ \mathsf{S}\right\|\right)$$

However, this type is still equivalent to $\mathsf{CLatticeStr}_2\ \mathsf{A}$. The intuitive reason is that if every subset S has (in the classical sense) a supremum and an infimum, then these are unique, which means we can constructively choose the supremum and infimum by using the principle of unique choice (Lemma 1.11.8). This intuitive argument can be formalized as follows.

We will show that the left components of $\mathsf{CLatticeStr}_3\ \mathsf{A}$ and $\mathsf{CLatticeStr}_2\ \mathsf{A}$ are equivalent, as the argument is similar for their right components. So, we want to show:

$$\left(\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \left\|\sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}\right\|\right) \simeq \left(\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}\right) \tag{2.1}$$

First, we show that for every $\mathsf{S} : \mathsf{L} \to \mathbf{Prop}$, type $\sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}$ is a mere proposition. Let $\mathsf{S} : \mathsf{L} \to \mathbf{Prop}$. Given $\mathsf{w}_1, \mathsf{w}_2 : \sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}$ we want to show $\mathsf{w}_1 = \mathsf{w}_2$. We already know that $\mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}$ is a mere proposition for every $\mathsf{w} : \mathsf{L}$. Therefore, by Lemma 1.8.14, it is enough to prove $\mathsf{pr}_1\ \mathsf{w}_1 = \mathsf{pr}_1\ \mathsf{w}_2$. But $(\mathsf{pr}_1\ \mathsf{w}_1)$ and $(\mathsf{pr}_1\ \mathsf{w}_2)$ are least upper bounds for S, which means that both $(\mathsf{pr}_1\ \mathsf{w}_1 \leq \mathsf{pr}_1\ \mathsf{w}_2)$ and $(\mathsf{pr}_1\ \mathsf{w}_2 \leq \mathsf{pr}_1\ \mathsf{w}_1)$ must be inhabited. Therefore, by antisymmetry, $\mathsf{pr}_1\ \mathsf{w}_1 = \mathsf{pr}_1\ \mathsf{w}_2$ is inhabited.

Hence, by Lemma 1.8.17, both sides of the equivalence (2.1) are mere propositions. Therefore, by Lemma 1.8.8, it is enough to prove that both sides are logically equivalent.

The right-to-left implication is obvious:

$$(\lambda \mathsf{f}.\ \lambda \mathsf{S}.\ |\mathsf{f}\ \mathsf{S}|) : \left(\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}\right) \to \left(\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \left\|\sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}\right\|\right)$$

For the left-to-right implication, suppose $\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \|\sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}\|$ is inhabited. We are going to use the principle of unique choice (Lemma 1.11.8) to conclude that $\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}$ must be inhabited.

We already know $\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \mathsf{IsProp}\ (\sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S})$ is inhabited, and $\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \|\sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLUB}\ \mathsf{w}\ \mathsf{S}\|$ is inhabited by hypothesis. Therefore, the required conclusion follows directly by the principle.

Hence, types $\mathsf{CLatticeStr}_3\ \mathsf{A}$ and $\mathsf{CLatticeStr}_2\ \mathsf{A}$ are equivalent, which means they are equivalent to $\mathsf{CLatticeStr}\ \mathsf{A}$.

Although type $\mathsf{CLatticeStr}_3\ \mathsf{A}$ is closer to the classical notion of complete lattice, on this report we will use type $\mathsf{CLatticeStr}\ \mathsf{A}$, since it is more "natural" from the point of view of type theory.

Now, we proceed to show some examples of complete lattices.

[4] Although it was not proved, Theorem 1.11.6 can be strengthened into an equivalence:

$$\left(\prod_{\mathsf{w}:\mathsf{T}} \sum_{\mathsf{a}:\mathsf{B}\ \mathsf{w}} \mathsf{P}\ \mathsf{w}\ \mathsf{a}\right) \simeq \left(\sum_{\mathsf{g}:\prod_{\mathsf{y}:\mathsf{T}} \mathsf{B}\ \mathsf{y}} \prod_{\mathsf{w}:\mathsf{T}} \mathsf{P}\ \mathsf{w}\ (\mathsf{g}\ \mathsf{w})\right)$$

The reader can find the proof for this stronger version at Theorem 2.15.7 in the HoTT book [20].

Example 2.2.7 (Complete Dual Lattice)**.** Let $\mathsf{A} :\equiv (\mathsf{L}, \leq, \mathsf{r}_1, \mathsf{a}_1, \mathsf{t}_1; \mathsf{sup}, \mathsf{sp}_1, \mathsf{inf}, \mathsf{ip}_1)$ be a complete lattice. Let $\mathsf{P} :\equiv (\mathsf{L}, \leq^{-1}, \mathsf{r}_2, \mathsf{a}_2, \mathsf{t}_2)$ be its dual *poset*, as constructed in Example 2.1.3 (interpreting A as a poset).

Then, we can augment P into a complete lattice by interchanging the role of functions sup and inf in A. This way, we will get the dual lattice for A. So, define the dual of A to be:

$$\mathsf{A}^{-1} :\equiv (\mathsf{L}, \leq^{-1}, \mathsf{r}_2, \mathsf{a}_2, \mathsf{t}_2; \mathsf{inf}, \mathsf{sp}_2, \mathsf{sup}, \mathsf{ip}_2)$$

where $\mathsf{sp}_2 : \prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \mathsf{IsLUB}^{\leq^{-1}}\ (\mathsf{inf}\ \mathsf{S})\ \mathsf{S}$ and $\mathsf{ip}_2 : \prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \mathsf{IsGLB}^{\leq^{-1}}\ (\mathsf{sup}\ \mathsf{S})\ \mathsf{S}$ are terms we still need to construct. As always, we only need to show that these types are inhabited.

- Type $\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \mathsf{IsLUB}^{\leq^{-1}}\ (\mathsf{inf}\ \mathsf{S})\ \mathsf{S}$ is inhabited.
  Let $\mathsf{S} : \mathsf{L} \to \mathbf{Prop}$.
  - First we need to show $\mathsf{IsUpperBound}^{\leq^{-1}}\ (\mathsf{inf}\ \mathsf{S})\ \mathsf{S}$ is inhabited.
    Let $\mathsf{y} : \mathsf{L}$ with $\mathsf{h}_1 : \mathsf{S}\ \mathsf{y}$. We need to show that $\mathsf{y} \leq^{-1} (\mathsf{inf}\ \mathsf{S})$ holds.
    By definition, $(\mathsf{y} \leq^{-1} (\mathsf{inf}\ \mathsf{S})) \equiv ((\mathsf{inf}\ \mathsf{S}) \leq \mathsf{y})$. But we have a proof:
    $$\mathsf{ip}_1 : \prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \mathsf{IsGLB}^{\leq}\ (\mathsf{inf}\ \mathsf{S})\ \mathsf{S}$$
    So, $\mathsf{pr}_1\ (\mathsf{ip}_1\ \mathsf{S})\ \mathsf{y}\ \mathsf{h}_1 : (\mathsf{inf}\ \mathsf{S}) \leq \mathsf{y}$. Therefore, $\mathsf{y} \leq^{-1} (\mathsf{inf}\ \mathsf{S})$ is inhabited.
  - Second, we need to show that the following type is inhabited:
    $$\prod_{\mathsf{y}:\mathsf{L}} (\mathsf{IsUpperBound}^{\leq^{-1}}\ \mathsf{y}\ \mathsf{S}) \to ((\mathsf{inf}\ \mathsf{S}) \leq^{-1} \mathsf{y})$$
    Let $\mathsf{y} : \mathsf{L}$ with $\mathsf{h}_1 : \mathsf{IsUpperBound}^{\leq^{-1}}\ \mathsf{y}\ \mathsf{S}$, or equivalently:
    $$\mathsf{h}_1 : \prod_{\mathsf{z}:\mathsf{L}} (\mathsf{S}\ \mathsf{z}) \to (\mathsf{z} \leq^{-1} \mathsf{y})$$
    Claim 1: Type $\mathsf{IsLowerBound}^{\leq}\ \mathsf{y}\ \mathsf{S}$ is inhabited.
    Let $\mathsf{z} : \mathsf{L}$ with $\mathsf{g}_1 : \mathsf{S}\ \mathsf{z}$. Then, $\mathsf{h}_1\ \mathsf{z}\ \mathsf{g}_1 : \mathsf{z} \leq^{-1} \mathsf{y}$. But $(\mathsf{z} \leq^{-1} \mathsf{y}) \equiv (\mathsf{y} \leq \mathsf{z})$.
    This proves the claim.
    Now, by definition, $((\mathsf{inf}\ \mathsf{S}) \leq^{-1} \mathsf{y}) \equiv (\mathsf{y} \leq (\mathsf{inf}\ \mathsf{S}))$. But we have a proof:
    $$\mathsf{ip}_1 : \prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \mathsf{IsGLB}^{\leq}\ (\mathsf{inf}\ \mathsf{S})\ \mathsf{S}$$
    By Claim 1, we have a term c of type $\mathsf{IsLowerBound}^{\leq}\ \mathsf{y}\ \mathsf{S}$. So:
    $$\mathsf{pr}_2\ (\mathsf{ip}_1\ \mathsf{S})\ \mathsf{y}\ \mathsf{c} : \mathsf{y} \leq (\mathsf{inf}\ \mathsf{S})$$
    Therefore, $(\mathsf{inf}\ \mathsf{S}) \leq^{-1} \mathsf{y}$ is inhabited.
- Type $\prod_{\mathsf{S}:\mathsf{L}\to\mathbf{Prop}} \mathsf{IsGLB}^{\leq^{-1}}\ (\mathsf{sup}\ \mathsf{S})\ \mathsf{S}$ is inhabited.
  This is proved like the previous case. Just carefully follow the definitions and interchange lower bounds and upper bounds, and infima and suprema from the previous proof.

▲

The next example extends the poset in Example 2.1.5 into a complete lattice. This will be a case where the propositional resizing axiom is required (see Section 1.8.1).

Example 2.2.8 ("Powertype" Complete Lattice)**.** Let $\mathsf{T} : \mathcal{U}$ be a type. Let $\mathsf{A} :\equiv (\mathsf{P}, \subseteq, \mathsf{r}, \mathsf{a}, \mathsf{t})$ be the "Powertype" poset for T as constructed in Example 2.1.5, where $\mathsf{P} :\equiv \mathsf{T} \to \mathbf{Prop}$, and $\subseteq$ is the subtype "inclusion" relation:

$$\subseteq :\equiv \left(\lambda \mathsf{A}.\, \lambda \mathsf{B}.\, \prod_{\mathsf{w}:\mathsf{T}} (\mathsf{A}\ \mathsf{w}) \to (\mathsf{B}\ \mathsf{w}) \right) : \mathsf{P} \to \mathsf{P} \to \mathbf{Prop}$$

We want to extend poset A into a complete lattice. For that matter, we need to provide a supremum function $\mathsf{sup} : (\mathsf{P} \to \mathbf{Prop}) \to \mathsf{P}$ and an infimum function $\mathsf{inf} : (\mathsf{P} \to \mathbf{Prop}) \to \mathsf{P}$.

In classical set theory, given a set T and a family $F \subseteq \mathcal{P}(T)$ of subsets of T, the supremum and infimum of F are:

$$\bigcup F = \{x \in T \mid \exists B \in F,\ x \in B\}$$
$$\bigcap F = \{x \in T \mid \forall B \in F,\ x \in B\}$$

In HoTT, since subtypes of T can be identified with predicates on T, the previous classical definitions are *tentatively* translated as follows. Given a family $\mathsf{F} : (\mathsf{T} \to \mathbf{Prop}) \to \mathbf{Prop}$ of subtypes of T, its supremum will be the subtype:

$$\left(\lambda \mathsf{w} : \mathsf{T}.\, \left\| \sum_{\mathsf{B}:\mathsf{T}\to\mathbf{Prop}} (\mathsf{F}\ \mathsf{B}) \times (\mathsf{B}\ \mathsf{w}) \right\| \right) : \mathsf{T} \to \mathbf{Prop} \tag{2.2}$$

which reads: "It is the subtype of all $\mathsf{w} : \mathsf{T}$ such that *there is* a subtype B of T belonging to the family F so that w is in B". Observe that we need to truncate the Σ-type to a mere proposition, as this will ensure that the lambda output will be a proposition (the Σ-type by itself is not necessarily a mere proposition, because $\mathsf{T} \to \mathbf{Prop}$ is a set, see Lemma 1.8.18).

Similarly, the infimum of F will be the subtype:

$$\left(\lambda \mathsf{w} : \mathsf{T}.\, \prod_{\mathsf{B}:\mathsf{T}\to\mathbf{Prop}} (\mathsf{F}\ \mathsf{B}) \to (\mathsf{B}\ \mathsf{w}) \right) : \mathsf{T} \to \mathbf{Prop} \tag{2.3}$$

which reads: "It is the subtype of all $\mathsf{w} : \mathsf{T}$ such that *any* subtype B of T belonging to the family F has w as an element". In this case, there is no need to truncate, because $\mathsf{B}\ \mathsf{w}$ is a mere proposition, and propositions are closed under Π-types (Lemma 1.8.17).

Although lambdas (2.2) and (2.3) seem to be well-typed, they are *not*. The reason is very subtle, and it has to do with universe levels. At the start of this chapter we claimed that most of the time we do not need to worry about universe levels, except for some "edge cases". This is one of those cases, as we are doing *impredicative* definitions. We are trying to construct two lambdas of type $\mathsf{T} \to \mathbf{Prop}$, both of which are quantifying over type $\mathsf{T} \to \mathbf{Prop}$ inside their own definition.

If we carelessly continue to use both lambdas as defined in (2.2) and (2.3), we may be causing an inconsistency in the theory. It is always a good idea to check all definitions in an automated proof assistant. For example, if we use Coq and try to define both lambdas as above, Coq will respond with a "universe inconsistency" error, which means that Coq was unable to assign a universe level to the types of both lambdas.

Therefore, it is a good rule of thumb to check universe levels whenever an impredicative definition is made. Let us check universe levels on both lambdas above. We will see that the propositional resizing axiom is needed to properly define them.

Let us put back universe levels on lambdas (2.2) and (2.3):

$$\left(\lambda \mathsf{w} : \mathsf{T}.\, \left\| \sum_{\mathsf{B}:\mathsf{T}\to\mathbf{Prop}_i} (\mathsf{F}\ \mathsf{B}) \times (\mathsf{B}\ \mathsf{w}) \right\| \right) : \mathsf{T} \to \mathbf{Prop}_i \tag{2.4}$$

$$\left(\lambda \mathsf{w} : \mathsf{T}.\, \prod_{\mathsf{B}:\mathsf{T}\to\mathbf{Prop}_i} (\mathsf{F}\ \mathsf{B}) \to (\mathsf{B}\ \mathsf{w}) \right) : \mathsf{T} \to \mathbf{Prop}_i \tag{2.5}$$

for any $\mathsf{F} : (\mathsf{T} \to \mathbf{Prop}_i) \to \mathbf{Prop}_j$ and $\mathsf{T} : \mathcal{U}_k$. The natural numbers i, j, and k represent arbitrary universe levels.

Although each appearance of $\mathbf{Prop}$ could be assigned a different level, there are certain restrictions forcing some levels to appear more than once. For example, level i must be assigned as specified because:

- The supremum function takes a family F with type $(\mathsf{T} \to \mathbf{Prop}_i) \to \mathbf{Prop}_j$ and must return something with type $\mathsf{T} \to \mathbf{Prop}_i$. Remember that we are defining a complete lattice structure on type $\mathsf{T} \to \mathbf{Prop}_i$, hence the universe level on the output type of the supremum function gets forced as specified. The same reasoning applies to the infimum function.
- In the Σ-type, B must have type $\mathsf{T} \to \mathbf{Prop}_i$ because F is applied on B. The same reasoning applies to the Π-type.

As there are no restrictions for levels j and k, they can be different from i.

Now, by Lemma 1.8.18, we know $\mathbf{Prop}_i$ is at level $i+1$. Therefore, $\mathsf{T} \to \mathbf{Prop}_i$ will be at level $\max\{k, i+1\}$ (see Remark 1.5.16). Also, $(\mathsf{F}\ \mathsf{B})$ is at level j and $(\mathsf{B}\ \mathsf{w})$ is at level i. This means $(\mathsf{F}\ \mathsf{B}) \times (\mathsf{B}\ \mathsf{w})$ and $(\mathsf{F}\ \mathsf{B}) \to (\mathsf{B}\ \mathsf{w})$ are at level $\max\{j, i\}$ (see Remark 1.5.29). This implies:

$$\left\| \sum_{\mathsf{B}:\mathsf{T}\to\mathbf{Prop}_i} (\mathsf{F}\ \mathsf{B}) \times (\mathsf{B}\ \mathsf{w}) \right\| : \mathcal{U}_{\max\{k,i+1,j\}} \tag{2.6}$$

and:

$$\left(\prod_{\mathsf{B}:\mathsf{T}\to\mathbf{Prop}_i} (\mathsf{F}\ \mathsf{B}) \to (\mathsf{B}\ \mathsf{w}) \right) : \mathcal{U}_{\max\{k,i+1,j\}} \tag{2.7}$$

But $\max\{k, i+1, j\} > i$, for otherwise we would have $i+1 \leq \max\{k, i+1, j\} \leq i$, which is impossible. Therefore, propositions (2.6) and (2.7) cannot be assigned into level i (i.e. they do not have type $\mathbf{Prop}_i$, as is required). We need some way of "moving the level down", and this is what the propositional resizing axiom is used for.

By Lemma 1.8.22 (which is a direct consequence of the propositional resizing axiom), there is an equivalence:

$$\mathsf{GPropR}_i^{\max\{k,i+1,j\}} : \mathbf{Prop}_i \to \mathbf{Prop}_{\max\{k,i+1,j\}}$$

With this equivalence, we can rewrite lambdas (2.4) and (2.5) as:

$$\left(\lambda \mathsf{w}:\mathsf{T}.\ \left(\mathsf{GPropR}_i^{\max\{k,i+1,j\}}\right)^{-1} \left\| \sum_{\mathsf{B}:\mathsf{T}\to\mathbf{Prop}_i} (\mathsf{F}\ \mathsf{B}) \times (\mathsf{B}\ \mathsf{w}) \right\| \right) : \mathsf{T} \to \mathbf{Prop}_i \tag{2.8}$$

$$\left(\lambda \mathsf{w}:\mathsf{T}.\ \left(\mathsf{GPropR}_i^{\max\{k,i+1,j\}}\right)^{-1} \left(\prod_{\mathsf{B}:\mathsf{T}\to\mathbf{Prop}_i} (\mathsf{F}\ \mathsf{B}) \to (\mathsf{B}\ \mathsf{w}) \right) \right) : \mathsf{T} \to \mathbf{Prop}_i \tag{2.9}$$

which are now correctly typed.

We are ready to define the supremum and infimum functions. By omitting universe levels in (2.8) and (2.9) and making use of definition $\mathsf{P} :\equiv \mathsf{T} \to \mathbf{Prop}$, the supremum and infimum functions can be defined as follows:

$$\mathsf{sup} :\equiv \left(\lambda \mathsf{F}.\ \lambda \mathsf{w}.\ \mathsf{GPropR}^{-1} \left\| \sum_{\mathsf{B}:\mathsf{P}} (\mathsf{F}\ \mathsf{B}) \times (\mathsf{B}\ \mathsf{w}) \right\| \right) : (\mathsf{P} \to \mathbf{Prop}) \to \mathsf{P}$$

$$\mathsf{inf} :\equiv \left(\lambda \mathsf{F}.\ \lambda \mathsf{w}.\ \mathsf{GPropR}^{-1} \left(\prod_{\mathsf{B}:\mathsf{P}} (\mathsf{F}\ \mathsf{B}) \to (\mathsf{B}\ \mathsf{w}) \right) \right) : (\mathsf{P} \to \mathbf{Prop}) \to \mathsf{P}$$

To finish this example, it remains to prove that the following types are inhabited:

$$\prod_{\mathsf{F}:\mathsf{P}\to\mathbf{Prop}} \mathsf{IsLUB}\ (\mathsf{sup}\ \mathsf{F})\ \mathsf{F}$$

$$\prod_{\mathsf{F}:\mathsf{P}\to\mathbf{Prop}} \mathsf{IsGLB}\ (\mathsf{inf}\ \mathsf{F})\ \mathsf{F}$$

- Type $\prod_{\mathsf{F}:\mathsf{P}\to\mathbf{Prop}} \mathsf{IsLUB}\ (\mathsf{sup}\ \mathsf{F})\ \mathsf{F}$ is inhabited.
  Let $\mathsf{F} : \mathsf{P} \to \mathbf{Prop}$.

– First, we need to show $\mathsf{IsUpperBound}\ (\mathsf{sup}\ \mathsf{F})\ \mathsf{F}$ is inhabited.
Let $\mathsf{E} : \mathsf{P}$ with $\mathsf{h}_1 : \mathsf{F}\ \mathsf{E}$. We need to show that $\mathsf{E} \subseteq (\mathsf{sup}\ \mathsf{F})$ holds, or equivalently, that the following type is inhabited:

$$\prod_{\mathsf{w}:\mathsf{T}} (\mathsf{E}\ \mathsf{w}) \to (\mathsf{sup}\ \mathsf{F}\ \mathsf{w})$$

So, let $\mathsf{w} : \mathsf{T}$ with $\mathsf{p}_1 : \mathsf{E}\ \mathsf{w}$. By definition of sup, we need to construct a term of type:

$$\mathsf{GPropR}^{-1} \left\| \sum_{\mathsf{B}:\mathsf{P}} (\mathsf{F}\ \mathsf{B}) \times (\mathsf{B}\ \mathsf{w}) \right\|$$

By Lemma 1.8.23, it is enough to build a term of type $\| \sum_{\mathsf{B}:\mathsf{P}} (\mathsf{F}\ \mathsf{B}) \times (\mathsf{B}\ \mathsf{w}) \|$.
So, consider term $|(\mathsf{E}, (\mathsf{h}_1, \mathsf{p}_1))|$, which has the correct type.

– Second, we need to show $\prod_{\mathsf{E}:\mathsf{P}} (\mathsf{IsUpperBound}\ \mathsf{E}\ \mathsf{F}) \to ((\mathsf{sup}\ \mathsf{F}) \subseteq \mathsf{E})$ is inhabited.
Let $\mathsf{E} : \mathsf{P}$ with $\mathsf{h}_1 : \mathsf{IsUpperBound}\ \mathsf{E}\ \mathsf{F}$, or equivalently:

$$\mathsf{h}_1 : \prod_{\mathsf{D}:\mathsf{P}} (\mathsf{F}\ \mathsf{D}) \to (\mathsf{D} \subseteq \mathsf{E})$$

By definition of $\subseteq$, we need to show that $\prod_{\mathsf{w}:\mathsf{T}} (\mathsf{sup}\ \mathsf{F}\ \mathsf{w}) \to (\mathsf{E}\ \mathsf{w})$ is inhabited, or equivalently, by definition of sup:

$$\prod_{\mathsf{w}:\mathsf{T}} \left(\mathsf{GPropR}^{-1} \left\| \sum_{\mathsf{B}:\mathsf{P}} (\mathsf{F}\ \mathsf{B}) \times (\mathsf{B}\ \mathsf{w}) \right\| \right) \to (\mathsf{E}\ \mathsf{w})$$

So, let $\mathsf{w} : \mathsf{T}$. By Lemma 1.8.24, it is enough to prove:

$$\left\| \sum_{\mathsf{B}:\mathsf{P}} (\mathsf{F}\ \mathsf{B}) \times (\mathsf{B}\ \mathsf{w}) \right\| \to (\mathsf{E}\ \mathsf{w})$$

but then, by (-1)-truncation recursion (Corollary 1.11.4) and Σ-recursion (Lemma 1.5.25) applied twice, it is enough to prove:

$$\prod_{\mathsf{B}:\mathsf{P}} ((\mathsf{F}\ \mathsf{B}) \to (\mathsf{B}\ \mathsf{w}) \to (\mathsf{E}\ \mathsf{w}))$$

Let $\mathsf{B} : \mathsf{P}$ with $\mathsf{p}_1 : \mathsf{F}\ \mathsf{B}$ and $\mathsf{p}_2 : \mathsf{B}\ \mathsf{w}$.
By hypothesis $\mathsf{h}_1 : \prod_{\mathsf{D}:\mathsf{P}} (\mathsf{F}\ \mathsf{D}) \to (\mathsf{D} \subseteq \mathsf{E})$. Therefore, $\mathsf{h}_1\ \mathsf{B}\ \mathsf{p}_1 : \mathsf{B} \subseteq \mathsf{E}$, or equivalently:

$$\mathsf{h}_1\ \mathsf{B}\ \mathsf{p}_1 : \prod_{\mathsf{y}:\mathsf{T}} (\mathsf{B}\ \mathsf{y}) \to (\mathsf{E}\ \mathsf{y})$$

which implies:

$$\mathsf{h}_1\ \mathsf{B}\ \mathsf{p}_1\ \mathsf{w}\ \mathsf{p}_2 : \mathsf{E}\ \mathsf{w}$$

- Type $\prod_{\mathsf{F}:\mathsf{P}\to\mathbf{Prop}} \mathsf{IsGLB}\ (\mathsf{inf}\ \mathsf{F})\ \mathsf{F}$ is inhabited.
The proof details are left to the reader. The proof strategy is similar to the previous case.

▲

Regarding monotone functions, we know that a complete lattice is also a poset, which means that monotone functions can also be considered to be defined on complete lattices. Also, we have the following corollary regarding monotone functions and complete dual lattices.

Corollary 2.2.9. *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \ldots)$ *be a complete lattice, and* $\mathsf{f} : \mathsf{L} \to \mathsf{L}$ *a monotone function.*
Let $\mathsf{A}^{-1} :\equiv (\mathsf{L}, \leq^{-1}, \ldots)$ *be the complete dual lattice of* A*, as constructed in Example 2.2.7. Then,* f *is monotone in* A^{-1}*, i.e. the following type is inhabited:*

$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{L}} (\mathsf{w} \leq^{-1} \mathsf{y}) \to (\mathsf{f}\ \mathsf{w} \leq^{-1} \mathsf{f}\ \mathsf{y})$$

Proof. By Lemma 2.1.8, we know f is monotone in the dual poset of A (interpreting A as a poset). Therefore, type $\prod_{\mathsf{w},\mathsf{y}:\mathsf{L}} (\mathsf{w} \leq^{-1} \mathsf{y}) \to (\mathsf{f}\,\mathsf{w} \leq^{-1} \mathsf{f}\,\mathsf{y})$ is inhabited, because this fact does not depend on the complete lattice components of A.

□

Now, we have a series of properties regarding suprema and infima for "two element sets". First, we introduce notation for such suprema and infima.

Definition 2.2.10 (Join and Meet)**.** Let $\mathsf{A} :\equiv (\mathsf{L}, \leq, \ldots; \mathsf{sup}, \ldots, \mathsf{inf}, \ldots)$ be a complete lattice, and $\mathsf{a}, \mathsf{b} : \mathsf{L}$ two elements in the lattice.

(i) Define:
$$\mathsf{a} \vee \mathsf{b} :\equiv \mathsf{sup}\ (\lambda \mathsf{w}:\mathsf{L}.\ \|(\mathsf{w} = \mathsf{a}) + (\mathsf{w} = \mathsf{b})\|)$$
to be the supremum of the subset $\lambda \mathsf{w}:\mathsf{L}.\ \|(\mathsf{w} = \mathsf{a}) + (\mathsf{w} = \mathsf{b})\| : \mathsf{L} \to \mathbf{Prop}$. This operation is called the *join* of a and b.

(ii) Define:
$$\mathsf{a} \wedge \mathsf{b} :\equiv \mathsf{inf}\ (\lambda \mathsf{w}:\mathsf{L}.\ \|(\mathsf{w} = \mathsf{a}) + (\mathsf{w} = \mathsf{b})\|)$$
to be the infimum of the same subset of L. This operation is called the *meet* of a and b.

▲

In Definition 2.2.10, we were forced to (-1)-truncate the sum type, even though $\mathsf{w} = \mathsf{a}$ and $\mathsf{w} = \mathsf{b}$ are mere propositions.[5] Remember that mere propositions are not closed under sum types, see comments after the proof of Lemma 1.8.17.

As was explained in Remark 1.11.5, a (-1)-truncated sum type behaves like a classical "or". Therefore, $\mathsf{a} \vee \mathsf{b}$ and $\mathsf{a} \wedge \mathsf{b}$ represent the supremum and infimum of the classical two element set:

$$\{a, b\} = \{x \in L \mid (x = a) \vee (x = b)\}$$

Lemma 2.2.11. *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \ldots; \mathsf{sup}, \mathsf{sp}, \mathsf{inf}, \mathsf{ip})$ *be a complete lattice, and* $\mathsf{a}, \mathsf{b} : \mathsf{L}$ *elements in the lattice. Then, the following mere propositions are inhabited:*

(i) $(\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{a}$.

(ii) $\mathsf{a} \leq (\mathsf{a} \vee \mathsf{b})$.

(iii) $(\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{b}$.

(iv) $\mathsf{b} \leq (\mathsf{a} \vee \mathsf{b})$.

Proof. Define $\mathsf{G} :\equiv \lambda \mathsf{w}:\mathsf{L}.\ \|(\mathsf{w} = \mathsf{a}) + (\mathsf{w} = \mathsf{b})\|$.

(i) We know $\mathsf{G}\,\mathsf{a} \equiv \|(\mathsf{a} = \mathsf{a}) + (\mathsf{a} = \mathsf{b})\|$ is inhabited, because term $\mathsf{t} :\equiv |\mathsf{inl}\ (\mathsf{refl}_{\mathsf{L}}\ \mathsf{a})|$ has the correct type. This means that "a is in the two element set $\{a, b\}$".

Now, we have a term $\mathsf{pr}_1\ (\mathsf{ip}\ \mathsf{G}) : \mathsf{IsLowerBound}\ (\mathsf{a} \wedge \mathsf{b})\ \mathsf{G}$, since $\mathsf{a} \wedge \mathsf{b} \equiv \mathsf{inf}\ \mathsf{G}$. But:

$$\mathsf{IsLowerBound}\ (\mathsf{a} \wedge \mathsf{b})\ \mathsf{G} :\equiv \prod_{\mathsf{y}:\mathsf{L}} (\mathsf{G}\,\mathsf{y}) \to ((\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{y})$$

Therefore, $\mathsf{pr}_1\ (\mathsf{ip}\ \mathsf{G})\ \mathsf{a}\ \mathsf{t} : (\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{a}$ as required.

(ii) Let $\mathsf{A}^{-1} :\equiv (\mathsf{L}, \leq^{-1}, \ldots)$ be the dual lattice of A as constructed in Example 2.2.7. Then, by (i) applied with A^{-1}, we have a term $\mathsf{t} : (\mathsf{a} \wedge^{-1} \mathsf{b}) \leq^{-1} \mathsf{a}$, where $\mathsf{a} \wedge^{-1} \mathsf{b}$ is notation for emphasizing that $\wedge$ should be taken in the dual lattice.

But $\mathsf{a} \wedge^{-1} \mathsf{b} \equiv \mathsf{sup}\ \mathsf{G}$, since the infimum in the dual lattice is the supremum function in A. And $\mathsf{sup}\ \mathsf{G} \equiv \mathsf{a} \vee \mathsf{b}$ by definition. So, $\mathsf{t} : (\mathsf{a} \vee \mathsf{b}) \leq^{-1} \mathsf{a}$.

But $(\mathsf{a} \vee \mathsf{b}) \leq^{-1} \mathsf{a} \equiv \mathsf{a} \leq (\mathsf{a} \vee \mathsf{b})$ by definition. Therefore, $\mathsf{t} : \mathsf{a} \leq (\mathsf{a} \vee \mathsf{b})$ as required.

[5] They are mere propositions because they are identities on a set, see Lemma 1.8.17.

(iii) The proof is similar to (i). Here, $\mathsf{pr}_1\ (\mathsf{ip}\ \mathsf{G})\ \mathsf{b}\ \mathsf{t} : (\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{b}$ will provide the proof, where $\mathsf{t} :\equiv |\mathsf{inr}\ (\mathsf{refl}_{\mathsf{L}}\ \mathsf{b})| : \mathsf{G}\ \mathsf{b}$, or equivalently, $\mathsf{t} : \|(\mathsf{b} = \mathsf{a}) + (\mathsf{b} = \mathsf{b})\|$.

(iv) The proof is similar to (ii). Here, we need to apply (iii) with A^{-1}, and $(\mathsf{a} \wedge^{-1} \mathsf{b}) \leq^{-1} \mathsf{b}$ will be definitionally equal to $\mathsf{b} \leq (\mathsf{a} \vee \mathsf{b})$ by duality.

□

We have some preservation properties with respect to $\wedge$ and $\vee$.

Lemma 2.2.12. *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \mathsf{rp}, \mathsf{ap}, \mathsf{tp}; \mathsf{sup}, \mathsf{sp}, \mathsf{inf}, \mathsf{ip})$ *be a complete lattice, and* $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d} : \mathsf{L}$ *elements in the lattice. Then, the following mere propositions are inhabited:*

(i) $(\mathsf{a} \leq \mathsf{c}) \to (\mathsf{b} \leq \mathsf{d}) \to ((\mathsf{a} \wedge \mathsf{b}) \leq (\mathsf{c} \wedge \mathsf{d}))$.

(ii) $(\mathsf{a} \leq \mathsf{c}) \to (\mathsf{b} \leq \mathsf{d}) \to ((\mathsf{a} \vee \mathsf{b}) \leq (\mathsf{c} \vee \mathsf{d}))$.

Proof. Define:

$$\mathsf{F} :\equiv \lambda \mathsf{w} : \mathsf{L}.\ \|(\mathsf{w} = \mathsf{a}) + (\mathsf{w} = \mathsf{b})\|$$
$$\mathsf{G} :\equiv \lambda \mathsf{w} : \mathsf{L}.\ \|(\mathsf{w} = \mathsf{c}) + (\mathsf{w} = \mathsf{d})\|$$

(i) Let $\mathsf{h}_1 : \mathsf{a} \leq \mathsf{c}$ and $\mathsf{h}_2 : \mathsf{b} \leq \mathsf{d}$. We need to show that $\mathsf{a} \wedge \mathsf{b}$ is a lower bound to the "two element set $\{c, d\}$". From this, the result will follow since $\mathsf{c} \wedge \mathsf{d}$ is the greatest lower bound for such set.

Claim 1: $\mathsf{IsLowerBound}\ (\mathsf{a} \wedge \mathsf{b})\ \mathsf{G}$ is inhabited.

Let $\mathsf{w} : \mathsf{L}$. Since we are trying to prove $(\mathsf{G}\ \mathsf{w}) \to ((\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{w})$, where $(\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{w}$ is a mere proposition, we can apply (-1)-truncation recursion (Corollary 1.11.4) to reduce the proof to $((\mathsf{w} = \mathsf{c}) + (\mathsf{w} = \mathsf{d})) \to ((\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{w})$.

Now, by the recursion principle for sum types (see Lemma 1.5.36), it is enough to prove the following two cases:

- $(\mathsf{w} = \mathsf{c}) \to ((\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{w})$ is inhabited.

  Let $\mathsf{p} : \mathsf{w} = \mathsf{c}$. By Lemma 2.2.11(i) we have a term $\mathsf{t}_1 : (\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{a}$. By transitivity of the order relation we have $\mathsf{t}_2 :\equiv \mathsf{tp}\ (\mathsf{a} \wedge \mathsf{b})\ \mathsf{a}\ \mathsf{c}\ \mathsf{t}_1\ \mathsf{h}_1 : (\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{c}$. And so, $\mathsf{transport}^{(\lambda \mathsf{x}.\ (\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{x})}\ \mathsf{p}^{-1}\ \mathsf{t}_2 : (\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{w}$.[6]

- $(\mathsf{w} = \mathsf{d}) \to ((\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{w})$ is inhabited.

  This is similar to the previous case.

This proves the claim.

Hence, we have:

$$\mathsf{pr}_2\ (\mathsf{ip}\ \mathsf{G}) : \prod_{\mathsf{y}:\mathsf{L}} (\mathsf{IsLowerBound}\ \mathsf{y}\ \mathsf{G}) \to (\mathsf{y} \leq \mathsf{inf}\ \mathsf{G})$$

and by Claim 1, there is a term $\mathsf{t} : \mathsf{IsLowerBound}\ (\mathsf{a} \wedge \mathsf{b})\ \mathsf{G}$.

Therefore, $\mathsf{pr}_2\ (\mathsf{ip}\ \mathsf{G})\ (\mathsf{a} \wedge \mathsf{b})\ \mathsf{t} : (\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{inf}\ \mathsf{G}$, or equivalently, $(\mathsf{a} \wedge \mathsf{b}) \leq (\mathsf{c} \wedge \mathsf{d})$, since $\mathsf{inf}\ \mathsf{G} \equiv (\mathsf{c} \wedge \mathsf{d})$ by definition.

(ii) The argument goes by duality. Just apply the previous case with the dual lattice A^{-1}.

□

We also have idempotency properties.

Lemma 2.2.13. *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \mathsf{rp}, \mathsf{ap}, \mathsf{tp}; \mathsf{sup}, \mathsf{sp}, \mathsf{inf}, \mathsf{ip})$ *be a complete lattice, and* $\mathsf{a} : \mathsf{L}$ *an element in the lattice. Then, the following mere propositions are inhabited:*

(i) $(\mathsf{a} \wedge \mathsf{a}) = \mathsf{a}$.

(ii) $(\mathsf{a} \vee \mathsf{a}) = \mathsf{a}$.

[6] -1 denotes the symmetry operator, see Lemma 1.5.59. Also, transport denotes the transport operator, see Lemma 1.5.61.

Proof. Define $\mathsf{G} :\equiv \lambda \mathsf{w}\colon \mathsf{L}.\ \|(\mathsf{w} = \mathsf{a}) + (\mathsf{w} = \mathsf{a})\|$.

(i) By Lemma 2.2.11(i) we have a term $\mathsf{t}_1 \colon (\mathsf{a} \wedge \mathsf{a}) \leq \mathsf{a}$. Therefore, if we can prove that there is a term $\mathsf{t}_2 \colon \mathsf{a} \leq (\mathsf{a} \wedge \mathsf{a})$, by antisymmetry we will have $\mathsf{ap}\ (\mathsf{a} \wedge \mathsf{a})\ \mathsf{a}\ \mathsf{t}_1\ \mathsf{t}_2 \colon (\mathsf{a} \wedge \mathsf{a}) = \mathsf{a}$.

To prove that $\mathsf{a} \leq (\mathsf{a} \wedge \mathsf{a})$ is inhabited, we need to show that a is a lower bound to the "two element set $\{a, a\}$". From this, the result will follow since $\mathsf{a} \wedge \mathsf{a}$ is the greatest lower bound for such set.

Claim 1: $\mathsf{IsLowerBound}\ \mathsf{a}\ \mathsf{G}$ is inhabited.

Let $\mathsf{w}\colon \mathsf{L}$. Since we are trying to prove $(\mathsf{G}\ \mathsf{w}) \to (\mathsf{a} \leq \mathsf{w})$, where $\mathsf{a} \leq \mathsf{w}$ is a mere proposition, we can apply (-1)-truncation recursion (Corollary 1.11.4) to reduce the proof to $((\mathsf{w} = \mathsf{a}) + (\mathsf{w} = \mathsf{a})) \to (\mathsf{a} \leq \mathsf{w})$.

Now, by the recursion principle for sum types (Lemma 1.5.36), it is enough to prove the following case:

- $(\mathsf{w} = \mathsf{a}) \to (\mathsf{a} \leq \mathsf{w})$ is inhabited.

  Let $\mathsf{p}\colon \mathsf{w} = \mathsf{a}$. By reflexivity, there is a term $\mathsf{r}_1 :\equiv \mathsf{rp}\ \mathsf{a}\colon \mathsf{a} \leq \mathsf{a}$. And so, $\mathsf{transport}^{(\lambda \mathsf{z}.\ \mathsf{a} \leq \mathsf{z})}\ \mathsf{p}^{-1}\ \mathsf{r}_1 \colon \mathsf{a} \leq \mathsf{w}$.

This proves the claim.

Hence, we have:

$$\mathsf{pr}_2\ (\mathsf{ip}\ \mathsf{G}) \colon \prod_{\mathsf{y}:\mathsf{L}} (\mathsf{IsLowerBound}\ \mathsf{y}\ \mathsf{G}) \to (\mathsf{y} \leq \mathsf{inf}\ \mathsf{G})$$

and by Claim 1, there is a term $\mathsf{t}\colon \mathsf{IsLowerBound}\ \mathsf{a}\ \mathsf{G}$.

Therefore, $\mathsf{pr}_2\ (\mathsf{ip}\ \mathsf{G})\ \mathsf{a}\ \mathsf{t} \colon \mathsf{a} \leq \mathsf{inf}\ \mathsf{G}$, or equivalently, $\mathsf{a} \leq (\mathsf{a} \wedge \mathsf{a})$ is inhabited, as $\mathsf{inf}\ \mathsf{G} \equiv (\mathsf{a} \wedge \mathsf{a})$ by definition.

(ii) The argument goes by duality. Just apply the previous case with the dual lattice A^{-1}.

□

Lastly, we have a compatibility corollary.

Corollary 2.2.14. *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \mathsf{rp}, \mathsf{ap}, \mathsf{tp}; \ldots)$ *be a complete lattice, and* $\mathsf{a}, \mathsf{b}, \mathsf{c}, \mathsf{d}\colon \mathsf{L}$ *elements in the lattice. Then, the following mere propositions are inhabited:*

(i) $(\mathsf{a} \leq \mathsf{c}) \to (\mathsf{a} \leq \mathsf{d}) \to (\mathsf{a} \leq (\mathsf{c} \wedge \mathsf{d}))$.

(ii) $(\mathsf{a} \leq \mathsf{c}) \to (\mathsf{b} \leq \mathsf{c}) \to ((\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{c})$.

(iii) $(\mathsf{a} \leq \mathsf{c}) \to (\mathsf{a} \leq \mathsf{d}) \to (\mathsf{a} \leq (\mathsf{c} \vee \mathsf{d}))$.

(iv) $(\mathsf{a} \leq \mathsf{c}) \to (\mathsf{b} \leq \mathsf{c}) \to ((\mathsf{a} \vee \mathsf{b}) \leq \mathsf{c})$.

Proof. (i) Let $\mathsf{h}_1 \colon \mathsf{a} \leq \mathsf{c}$ and $\mathsf{h}_2 \colon \mathsf{a} \leq \mathsf{d}$.

By Lemma 2.2.12(i), there is a term:

$$\mathsf{p}_1 \colon (\mathsf{a} \leq \mathsf{c}) \to (\mathsf{a} \leq \mathsf{d}) \to ((\mathsf{a} \wedge \mathsf{a}) \leq (\mathsf{c} \wedge \mathsf{d}))$$

Therefore, $\mathsf{p}_2 :\equiv \mathsf{p}_1\ \mathsf{h}_1\ \mathsf{h}_2 \colon (\mathsf{a} \wedge \mathsf{a}) \leq (\mathsf{c} \wedge \mathsf{d})$.

By Lemma 2.2.13(i), there is a term $\mathsf{q}\colon (\mathsf{a} \wedge \mathsf{a}) = \mathsf{a}$. Therefore:

$$\mathsf{transport}^{(\lambda \mathsf{z}.\ \mathsf{z} \leq (\mathsf{c} \wedge \mathsf{d}))}\ \mathsf{q}\ \mathsf{p}_2 \colon \mathsf{a} \leq (\mathsf{c} \wedge \mathsf{d})$$

(ii) Let $\mathsf{h}_1 \colon \mathsf{a} \leq \mathsf{c}$ and $\mathsf{h}_2 \colon \mathsf{b} \leq \mathsf{c}$.

By Lemma 2.2.11(i), there is a term $\mathsf{p}_1 \colon (\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{a}$. Therefore, by transitivity of the order relation, we have $\mathsf{tp}\ (\mathsf{a} \wedge \mathsf{b})\ \mathsf{a}\ \mathsf{c}\ \mathsf{p}_1\ \mathsf{h}_1 \colon (\mathsf{a} \wedge \mathsf{b}) \leq \mathsf{c}$ as required.

(iii) This is similar to (ii).

(iv) This is similar to (i). □

2.3 Fixpoints and the Knaster-Tarski Lemma

We proceed to prove the main result on this chapter, the so-called Knaster-Tarski Lemma, which obtains fixpoints from monotone functions. For that matter, we need to define a couple of concepts. Although these concepts are defined relative to posets, they also apply to complete lattices by Convention 2.2.6.

Definition 2.3.1 (Prefixpoint, postfixpoint and fixpoint)**.** Let $\mathsf{A} :\equiv (\mathsf{P}, \leq, \ldots)$ be a poset, $\mathsf{f} : \mathsf{P} \to \mathsf{P}$ a function, and $\mathsf{w} : \mathsf{P}$ an element in the poset.

(i) If the following proposition is inhabited:

$$\mathsf{IsPrefixpoint}\ \mathsf{w}\ \mathsf{f} :\equiv \mathsf{f}\ \mathsf{w} \leq \mathsf{w}$$

we say that w is a *prefixpoint* of f.

(ii) If the following proposition is inhabited:

$$\mathsf{IsPostfixpoint}\ \mathsf{w}\ \mathsf{f} :\equiv \mathsf{w} \leq \mathsf{f}\ \mathsf{w}$$

we say that w is a *postfixpoint* of f.

(iii) If the following proposition is inhabited:

$$\mathsf{IsFixpoint}\ \mathsf{w}\ \mathsf{f} :\equiv \mathsf{f}\ \mathsf{w} = \mathsf{w}$$

we say that w is a *fixpoint* of f.

▲

Convention 2.3.2. When the order relation $\leq$ is not clear from context, we will emphasize it as $(\mathsf{IsPrefixpoint}^{\leq}\ \mathsf{w}\ \mathsf{f})$ and $(\mathsf{IsPostfixpoint}^{\leq}\ \mathsf{w}\ \mathsf{f})$, written in superscript.

▲

Fixpoints were defined independently of prefixpoints and postfixpoints, but fixpoints are those elements that are prefixpoints and postfixpoints at the same time, as the following lemma shows.

Lemma 2.3.3. *Let* $\mathsf{A} :\equiv (\mathsf{P}, \leq, \mathsf{rp}, \mathsf{ap}, \mathsf{tp})$ *be a poset,* $\mathsf{f} : \mathsf{P} \to \mathsf{P}$ *a function, and* $\mathsf{w} : \mathsf{P}$ *an element in the poset. Then, the following mere propositions are equivalent:*

- $\mathsf{IsFixpoint}\ \mathsf{w}\ \mathsf{f}$.
- $(\mathsf{IsPrefixpoint}\ \mathsf{w}\ \mathsf{f}) \times (\mathsf{IsPostfixpoint}\ \mathsf{w}\ \mathsf{f})$

Proof. To prove that two mere propositions are equivalent, it is enough to prove they are *logically equivalent* (Lemma 1.8.8). So, it is enough to prove the following two implications:

- $(\mathsf{IsFixpoint}\ \mathsf{w}\ \mathsf{f}) \to ((\mathsf{IsPrefixpoint}\ \mathsf{w}\ \mathsf{f}) \times (\mathsf{IsPostfixpoint}\ \mathsf{w}\ \mathsf{f}))$.

  Let $\mathsf{h} : \mathsf{IsFixpoint}\ \mathsf{w}\ \mathsf{f}$, or equivalently, $\mathsf{h} : \mathsf{f}\ \mathsf{w} = \mathsf{w}$.

  By reflexivity of the order relation, we have $\mathsf{t} :\equiv \mathsf{rp}\ (\mathsf{f}\ \mathsf{w}) : \mathsf{f}\ \mathsf{w} \leq \mathsf{f}\ \mathsf{w}$. So:

$$\mathsf{t}_1 :\equiv (\mathsf{transport}^{(\lambda \mathsf{z}.\ (\mathsf{f}\ \mathsf{w}) \leq \mathsf{z})}\ \mathsf{h}\ \mathsf{t}) : \mathsf{f}\ \mathsf{w} \leq \mathsf{w}$$

  and:

$$\mathsf{t}_2 :\equiv (\mathsf{transport}^{(\lambda \mathsf{z}.\ \mathsf{z} \leq (\mathsf{f}\ \mathsf{w}))}\ \mathsf{h}\ \mathsf{t}) : \mathsf{w} \leq \mathsf{f}\ \mathsf{w}$$

  Therefore, $(\mathsf{t}_1, \mathsf{t}_2) : (\mathsf{IsPrefixpoint}\ \mathsf{w}\ \mathsf{f}) \times (\mathsf{IsPostfixpoint}\ \mathsf{w}\ \mathsf{f})$.

- $((\mathsf{IsPrefixpoint}\ \mathsf{w}\ \mathsf{f}) \times (\mathsf{IsPostfixpoint}\ \mathsf{w}\ \mathsf{f})) \to (\mathsf{IsFixpoint}\ \mathsf{w}\ \mathsf{f})$.

  By the recursion principle for Σ-types, it is enough to prove:

  $$(\mathsf{IsPrefixpoint}\ \mathsf{w}\ \mathsf{f}) \to (\mathsf{IsPostfixpoint}\ \mathsf{w}\ \mathsf{f}) \to (\mathsf{IsFixpoint}\ \mathsf{w}\ \mathsf{f})$$

  Let $\mathsf{h}_1 : \mathsf{IsPrefixpoint}\ \mathsf{w}\ \mathsf{f}$ and $\mathsf{h}_2 : \mathsf{IsPostfixpoint}\ \mathsf{w}\ \mathsf{f}$, or equivalently:

  $$\mathsf{h}_1 : \mathsf{f}\ \mathsf{w} \leq \mathsf{w}$$
  $$\mathsf{h}_2 : \mathsf{w} \leq \mathsf{f}\ \mathsf{w}$$

  Then, by antisymmetry of the order relation, we have:

  $$\mathsf{ap}\ (\mathsf{f}\ \mathsf{w})\ \mathsf{w}\ \mathsf{h}_1\ \mathsf{h}_2 : \mathsf{f}\ \mathsf{w} = \mathsf{w}$$

□

Now, among the fixpoints, there are two special ones (if they exist at all): the greatest and the smallest.

Definition 2.3.4 (Least and greatest fixpoint)**.** Let $\mathsf{A} :\equiv (\mathsf{P}, \leq, \ldots)$ be a poset, $\mathsf{f} : \mathsf{P} \to \mathsf{P}$ a function, and $\mathsf{w} : \mathsf{P}$ an element in the poset.

(i) If the following type is inhabited:

$$\mathsf{IsLFP}\ \mathsf{w}\ \mathsf{f} :\equiv (\mathsf{IsFixpoint}\ \mathsf{w}\ \mathsf{f}) \times \left(\prod_{\mathsf{y}:\mathsf{P}} (\mathsf{IsFixpoint}\ \mathsf{y}\ \mathsf{f}) \to (\mathsf{w} \leq \mathsf{y}) \right)$$

we say that w is the *least fixpoint* for function f.

(ii) If the following type is inhabited:

$$\mathsf{IsGFP}\ \mathsf{w}\ \mathsf{f} :\equiv (\mathsf{IsFixpoint}\ \mathsf{w}\ \mathsf{f}) \times \left(\prod_{\mathsf{y}:\mathsf{P}} (\mathsf{IsFixpoint}\ \mathsf{y}\ \mathsf{f}) \to (\mathsf{y} \leq \mathsf{w}) \right)$$

we say that w is the *greatest fixpoint* for function f.

▲

Convention 2.3.5. When the order relation $\leq$ is not clear from context, we will emphasize it as $(\mathsf{IsLFP}^{\leq}\ \mathsf{w}\ \mathsf{f})$ and $(\mathsf{IsGFP}^{\leq}\ \mathsf{w}\ \mathsf{f})$, written in superscript.

▲

Both components in types $(\mathsf{IsLFP}\ \mathsf{w}\ \mathsf{f})$ and $(\mathsf{IsGFP}\ \mathsf{w}\ \mathsf{f})$ are mere propositions (propositions are closed under Σ-types and Π-types). $\mathsf{IsLFP}\ \mathsf{w}\ \mathsf{f}$ reads: "w is a fixpoint for function f, and any other fixpoint for f is above w". $\mathsf{IsGFP}\ \mathsf{w}\ \mathsf{f}$ reads: "w is a fixpoint for function f, and any other fixpoint for f is below w".

Also, notice we used the definite article in the definitions of $(\mathsf{IsLFP}\ \mathsf{w}\ \mathsf{f})$ and $(\mathsf{IsGFP}\ \mathsf{w}\ \mathsf{f})$ (i.e. we said *the* least fixpoint and *the* greatest fixpoint). This can be justified as follows. Let a and b be two least fixpoints for function f. Then, since both of them are fixpoints and least fixpoints, both $\mathsf{a} \leq \mathsf{b}$ and $\mathsf{b} \leq \mathsf{a}$ will hold. Therefore, by antisymmetry, $\mathsf{a} = \mathsf{b}$ will be inhabited. Hence, the least fixpoint is unique (the argument is similar for the greatest fixpoint).

We are ready to state the Knaster-Tarski Lemma.

Lemma 2.3.6 (Knaster-Tarski)**.** *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \mathsf{rp}, \mathsf{ap}, \mathsf{tp}; \mathsf{sup}, \mathsf{sp}, \mathsf{inf}, \mathsf{ip})$ *be a complete lattice, and* $\mathsf{f} : \mathsf{L} \to \mathsf{L}$ *a monotone function. Then,*

(i) f *has a least fixpoint in* L*, or equivalently, there is a pair:*

$$(\mathsf{Lfp}\ \mathsf{f},\ \mathsf{Lfp\_pr}\ \mathsf{f}) : \sum_{\mathsf{w}:\mathsf{L}} \mathsf{IsLFP}\ \mathsf{w}\ \mathsf{f}$$

where $\mathsf{Lfp}\ \mathsf{f}$ *denotes the least fixpoint of* f*, and* $\mathsf{Lfp\_pr}\ \mathsf{f}$ *denotes a proof that* $\mathsf{Lfp}\ \mathsf{f}$ *is the least fixpoint of* f*.*

(ii) f *has a greatest fixpoint in* L*, or equivalently, there is a pair:*

$$(\mathsf{Gfp}\ \mathsf{f},\ \mathsf{Gfp\_pr}\ \mathsf{f}) : \sum_{\mathsf{w:L}} \mathsf{IsGFP}\ \mathsf{w}\ \mathsf{f}$$

where $\mathsf{Gfp}\ \mathsf{f}$ *denotes the greatest fixpoint of* f*, and* $\mathsf{Gfp\_pr}\ \mathsf{f}$ *denotes a proof that* $\mathsf{Gfp}\ \mathsf{f}$ *is the greatest fixpoint of* f*.*

Proof. (i) Define:

$$\mathsf{P} :\equiv (\lambda \mathsf{w} : \mathsf{L}.\ \mathsf{IsPrefixpoint}\ \mathsf{w}\ \mathsf{f}) : \mathsf{L} \to \mathbf{Prop}$$

to be the subset of all elements in L that are prefixpoints of f.

Our candidate for the least fixpoint for f will be the infimum of P:

$$\mathsf{Lfp}\ \mathsf{f} :\equiv \mathsf{inf}\ \mathsf{P}$$

If we can build a term $\mathsf{Lfp\_pr}\ \mathsf{f} : \mathsf{IsLFP}\ (\mathsf{Lfp}\ \mathsf{f})\ \mathsf{f}$, then pair $(\mathsf{Lfp}\ \mathsf{f},\ \mathsf{Lfp\_pr}\ \mathsf{f}) : \sum_{\mathsf{w:L}} \mathsf{IsLFP}\ \mathsf{w}\ \mathsf{f}$ will be the required proof.

Term $\mathsf{Lfp\_pr}\ \mathsf{f}$ will exist if we can prove that $\mathsf{IsLFP}\ (\mathsf{Lfp}\ \mathsf{f})\ \mathsf{f}$ is inhabited. But first, we prove a claim.

Claim 1: Proposition $\mathsf{IsLowerBound}\ (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}))\ \mathsf{P}$ is inhabited.

Let $\mathsf{w} : \mathsf{L}$ with $\mathsf{h}_1 : \mathsf{P}\ \mathsf{w}$, or equivalently, $\mathsf{h}_1 : \mathsf{f}\ \mathsf{w} \leq \mathsf{w}$. By definition, $\mathsf{Lfp}\ \mathsf{f}$ is the infimum of P, which means $\mathsf{pr}_1\ (\mathsf{ip}\ \mathsf{P}) : \mathsf{IsLowerBound}\ (\mathsf{Lfp}\ \mathsf{f})\ \mathsf{P}$ (since $\mathsf{Lfp}\ \mathsf{f} \equiv \mathsf{inf}\ \mathsf{P}$). So, $\mathsf{p}_1 :\equiv \mathsf{pr}_1\ (\mathsf{ip}\ \mathsf{P})\ \mathsf{w}\ \mathsf{h}_1 : \mathsf{Lfp}\ \mathsf{f} \leq \mathsf{w}$.

But f is monotone, i.e. there is a term:

$$\mathsf{m} : \prod_{\mathsf{w,y:L}} (\mathsf{w} \leq \mathsf{y}) \to (\mathsf{f}\ \mathsf{w} \leq \mathsf{f}\ \mathsf{y})$$

which implies, $\mathsf{p}_2 :\equiv \mathsf{m}\ (\mathsf{Lfp}\ \mathsf{f})\ \mathsf{w}\ \mathsf{p}_1 : \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}) \leq \mathsf{f}\ \mathsf{w}$.

Hence, by transitivity:

$$\mathsf{tp}\ (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}))\ (\mathsf{f}\ \mathsf{w})\ \mathsf{w}\ \mathsf{p}_2\ \mathsf{h}_1 : \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}) \leq \mathsf{w}$$

This proves the claim.

Now, we proceed to prove $\mathsf{IsLFP}\ (\mathsf{Lfp}\ \mathsf{f})\ \mathsf{f}$ by showing the following two propositions:

- $\mathsf{IsFixpoint}\ (\mathsf{Lfp}\ \mathsf{f})\ \mathsf{f}$ is inhabited.

  By Claim 1, there is a term $\mathsf{p}_3 : \mathsf{IsLowerBound}\ (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}))\ \mathsf{P}$. By definition, $\mathsf{Lfp}\ \mathsf{f}$ is the greatest lower bound for P, so:

  $$\mathsf{p}_4 :\equiv \mathsf{pr}_2\ (\mathsf{ip}\ \mathsf{P})\ (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}))\ \mathsf{p}_3 : \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}) \leq \mathsf{inf}\ \mathsf{P}$$

  or, equivalently, $\mathsf{p}_4 : \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}) \leq \mathsf{Lfp}\ \mathsf{f}$.

  Therefore, if we can prove $\mathsf{Lfp}\ \mathsf{f} \leq \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f})$, then, by antisymmetry, $\mathsf{Lfp}\ \mathsf{f}$ will be a fixpoint.

  But f is monotone, i.e. there is a term $\mathsf{m} : \prod_{\mathsf{w,y:L}} (\mathsf{w} \leq \mathsf{y}) \to (\mathsf{f}\ \mathsf{w} \leq \mathsf{f}\ \mathsf{y})$. Therefore:

  $$\mathsf{p}_5 :\equiv \mathsf{m}\ (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}))\ (\mathsf{Lfp}\ \mathsf{f})\ \mathsf{p}_4 : \mathsf{f}\ (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f})) \leq \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f})$$

  or equivalently, $\mathsf{p}_5 : \mathsf{IsPrefixpoint}\ (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}))\ \mathsf{f}$.

  By definition of P, this can be restated as $\mathsf{p}_5 : \mathsf{P}\ (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}))$.

  But $\mathsf{Lfp}\ \mathsf{f}$ is a lower bound for P, so $\mathsf{pr}_1\ (\mathsf{ip}\ \mathsf{P})\ (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}))\ \mathsf{p}_5 : \mathsf{Lfp}\ \mathsf{f} \leq \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f})$ as required.

- $\prod_{\mathsf{y:L}} (\mathsf{IsFixpoint}\ \mathsf{y}\ \mathsf{f}) \to (\mathsf{Lfp}\ \mathsf{f} \leq \mathsf{y})$ is inhabited.

  Let $\mathsf{y} : \mathsf{L}$ with $\mathsf{h} : \mathsf{IsFixpoint}\ \mathsf{y}\ \mathsf{f}$, or equivalently, $\mathsf{h} : \mathsf{f}\ \mathsf{y} = \mathsf{y}$.

  By reflexivity, we have $\mathsf{p}_6 :\equiv \mathsf{rp}\ \mathsf{y} : \mathsf{y} \leq \mathsf{y}$.

  But then, $\mathsf{p}_7 :\equiv \mathsf{transport}^{(\lambda \mathsf{z}.\ \mathsf{z} \leq \mathsf{y})}\ \mathsf{h}^{-1}\ \mathsf{p}_6 : \mathsf{f}\ \mathsf{y} \leq \mathsf{y}$, or equivalently, $\mathsf{p}_7 : \mathsf{IsPrefixpoint}\ \mathsf{y}\ \mathsf{f}$. Hence, $\mathsf{p}_7 : \mathsf{P}\ \mathsf{y}$ by definition of P.

  But $\mathsf{Lfp}\ \mathsf{f}$ is a lower bound for P, so $\mathsf{pr}_1\ (\mathsf{ip}\ \mathsf{P})\ \mathsf{y}\ \mathsf{p}_7 : \mathsf{Lfp}\ \mathsf{f} \leq \mathsf{y}$ as required.

(ii) The argument goes by duality. Just apply the previous case on the dual lattice A^{-1}.

Notice that the least fixpoint in the dual lattice A^{-1} will correspond to the greatest fixpoint in lattice A. Also, the greatest fixpoint in lattice A will be:

$$\mathsf{sup}\ (\lambda \mathsf{w} : \mathsf{L}.\ \mathsf{IsPostfixpoint}^{\leq}\ \mathsf{w}\ \mathsf{f})$$

□

2.4 Induction and coinduction principles

In standard mathematics, the existence of "inductively" and "coinductively" defined sets is justified by the Knaster-Tarski Lemma, because these sets are implicitly constructed as a fixpoint of a monotone function in the powerset lattice (see [16] for examples).

Also, whenever we want to prove a property about these "inductively" and "coinductively" defined sets, we make use of proof techniques that take advantage of their inductive and coinductive nature. These proof techniques are the so-called *induction and coinduction principles*. These principles are corollaries to the Knaster-Tarski Lemma, as the following results show.

First, the induction principles.

Corollary 2.4.1 (Conventional induction principle)**.** *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \ldots; \ldots, \mathsf{inf}, \mathsf{ip})$ *be a complete lattice,* $\mathsf{f} : \mathsf{L} \to \mathsf{L}$ *a monotone function, and* $\mathsf{w} : \mathsf{L}$ *an element in the complete lattice. Then, the following mere proposition is inhabited:*[7]

$$(\mathsf{f}\ \mathsf{w} \leq \mathsf{w}) \to (\mathsf{Lfp}\ \mathsf{f} \leq \mathsf{w})$$

Proof. Let $\mathsf{h} : \mathsf{f}\ \mathsf{w} \leq \mathsf{w}$. In the proof of Lemma 2.3.6(i), $\mathsf{Lfp}\ \mathsf{f}$ was constructed as:

$$\mathsf{Lfp}\ \mathsf{f} :\equiv \mathsf{inf}\ \mathsf{P}$$

where:

$$\mathsf{P} :\equiv \lambda \mathsf{y} : \mathsf{L}.\ \mathsf{IsPrefixpoint}\ \mathsf{y}\ \mathsf{f}$$

Since $\mathsf{h} : \mathsf{f}\ \mathsf{w} \leq \mathsf{w}$, we have $\mathsf{h} : \mathsf{P}\ \mathsf{w}$. But $\mathsf{Lfp}\ \mathsf{f}$ is the infimum of P (and hence a lower bound for P). So, $\mathsf{pr}_1\ (\mathsf{ip}\ \mathsf{P})\ \mathsf{w}\ \mathsf{h} : \mathsf{Lfp}\ \mathsf{f} \leq \mathsf{w}$ as required.

□

To understand the statement of the conventional induction principle, it is useful to think of A as some particular complete lattice. In all the following explanations, A will denote the powertype lattice of Example 2.2.8. In other words, we have:

$$\mathsf{A} :\equiv (\mathsf{T} \to \mathbf{Prop}, \subseteq, \mathsf{rp}, \mathsf{ap}, \mathsf{tp}; \mathsf{sup}, \mathsf{sp}, \mathsf{inf}, \mathsf{ip})$$

where $\subseteq$ is the "inclusion" relation.

Monotone functions $\mathsf{f} : (\mathsf{T} \to \mathbf{Prop}) \to (\mathsf{T} \to \mathbf{Prop})$ on this lattice can be pictured as procedures that map subtypes of T into subtypes "closer" to a fixpoint of f, i.e. given a subtype $\mathsf{C} : \mathsf{T} \to \mathbf{Prop}$, then $\mathsf{f}\ \mathsf{C} : \mathsf{T} \to \mathbf{Prop}$ is a subtype closer to a fixpoint of f. Hence, fixpoints of f are subtypes that cannot be "improved further".

In this lattice, the conventional induction principle reads: "To prove that all elements of $\mathsf{Lfp}\ \mathsf{f}$ are in w (i.e. $\mathsf{Lfp}\ \mathsf{f} \subseteq \mathsf{w}$), prove that any improvement of w falls inside w again". In other words, *if* either w has points in excess (because the improvement falls inside w again) or w is already a fixpoint, *then* w *must* contain some fixpoint, and as a consequence, it will contain the least fixpoint.

The next induction principle states that it is not necessary to check that the improvement of *all* of w falls inside w again, but it is enough to take those elements in w belonging to the least fixpoint of f (i.e. condition $\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{w}$ in the following induction principle).

Corollary 2.4.2 (Extended conventional induction principle)**.** *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \mathsf{rp}, \mathsf{ap}, \mathsf{tp}; \ldots)$ *be a complete lattice,* $\mathsf{f} : \mathsf{L} \to \mathsf{L}$ *a monotone function, and* $\mathsf{w} : \mathsf{L}$ *an element in the complete lattice. Then, the following mere proposition is inhabited:*

$$(\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{w}) \to (\mathsf{Lfp}\ \mathsf{f} \leq \mathsf{w})$$

[7] $\mathsf{Lfp}\ \mathsf{f}$ exists by the Knaster-Tarski Lemma.

Proof. Let $\mathsf{h} : \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{w}$.

By Lemma 2.2.11(i) we have a term $\mathsf{t}_1 : (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{Lfp}\ \mathsf{f}$.

Since f is monotone (i.e. there is a term $\mathsf{m} : \prod_{\mathsf{w},\mathsf{y}:\mathsf{L}} (\mathsf{w} \leq \mathsf{y}) \to (\mathsf{f}\ \mathsf{w} \leq \mathsf{f}\ \mathsf{y})$), we have:

$$\mathsf{t}_2 :\equiv \mathsf{m}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w})\ (\mathsf{Lfp}\ \mathsf{f})\ \mathsf{t}_1 : \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f})$$

But $\mathsf{Lfp}\ \mathsf{f}$ is a fixpoint of f, i.e. there is $\mathsf{t}_3 :\equiv \mathsf{pr}_1\ (\mathsf{Lfp\_pr}\ \mathsf{f}) : \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f}) = \mathsf{Lfp}\ \mathsf{f}$. So:

$$\mathsf{t}_4 :\equiv \mathsf{transport}^{(\lambda \mathsf{z}.\ \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{z})}\ \mathsf{t}_3\ \mathsf{t}_2 : \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{Lfp}\ \mathsf{f}$$

Now, by Corollary 2.2.14(i), there is a term:

$$\mathsf{p} : (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{Lfp}\ \mathsf{f}) \to (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{w}) \to (\mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}))$$

So, there is a term $\mathsf{t}_5 :\equiv \mathsf{p}\ \mathsf{t}_4\ \mathsf{h} : \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w})$.

Term t_5 is a proof that $\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}$ is a prefixpoint of f. Therefore, by conventional induction, there is a term $\mathsf{t}_6 : \mathsf{Lfp}\ \mathsf{f} \leq (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w})$. But, by Lemma 2.2.11(iii), there is a term $\mathsf{t}_7 : (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{w}$. So, by transitivity of the order relation:

$$\mathsf{tp}\ (\mathsf{Lfp}\ \mathsf{f})\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w})\ \mathsf{w}\ \mathsf{t}_6\ \mathsf{t}_7 : \mathsf{Lfp}\ \mathsf{f} \leq \mathsf{w}$$

□

The following induction principle has an "iterative flavor" to it. Its core idea comes from Mendler-style recursion schemes (see [3] for extensive examples).

Corollary 2.4.3 (Mendler-style induction principle)**.** *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \mathsf{rp}, \mathsf{ap}, \mathsf{tp}; \ldots)$ *be a complete lattice,* $\mathsf{f} : \mathsf{L} \to \mathsf{L}$ *a monotone function, and* $\mathsf{w} : \mathsf{L}$ *an element in the complete lattice. Then, the following mere proposition is inhabited:*

$$\left(\prod_{\mathsf{y}:\mathsf{L}} (\mathsf{y} \leq \mathsf{w}) \to (\mathsf{f}\ \mathsf{y} \leq \mathsf{w})\right) \to (\mathsf{Lfp}\ \mathsf{f} \leq \mathsf{w})$$

Proof. Let $\mathsf{h} : \prod_{\mathsf{y}:\mathsf{L}} (\mathsf{y} \leq \mathsf{w}) \to (\mathsf{f}\ \mathsf{y} \leq \mathsf{w})$.

Then, $\mathsf{t}_1 :\equiv \mathsf{h}\ \mathsf{w}\ (\mathsf{rp}\ \mathsf{w}) : \mathsf{f}\ \mathsf{w} \leq \mathsf{w}$. So, t_1 is a proof that w is a prefixpoint of f. Therefore, by conventional induction, there is a term $\mathsf{t}_2 : \mathsf{Lfp}\ \mathsf{f} \leq \mathsf{w}$ as required.

□

In the powertype lattice, Mendler-style induction states: "To prove that all elements in $\mathsf{Lfp}\ \mathsf{f}$ are in w, prove that any subtype y contained in w has its improvement contained in w". The hypothesis (instantiated in the powertype lattice):

$$\prod_{\mathsf{y}:\mathsf{T} \to \mathbf{Prop}} (\mathsf{y} \subseteq \mathsf{w}) \to (\mathsf{f}\ \mathsf{y} \subseteq \mathsf{w}) \tag{2.10}$$

is implicitly stating an iterative condition, as the following explanation shows.

Suppose $\bot$ is the empty subtype:

$$\bot :\equiv (\lambda \mathsf{Y}.\ \mathbb{0}) : \mathsf{T} \to \mathbf{Prop}$$

where the empty type $\mathbb{0}$ is a proposition by Lemma 1.8.17.

By definition of the inclusion relation, we know:

$$(\bot \subseteq \mathsf{w}) \equiv \prod_{\mathsf{z}:\mathsf{T}} (\mathbb{0} \to (\mathsf{w}\ \mathsf{z}))$$

Therefore, by the recursion principle for the empty type (Lemma 1.5.42), we have that $\bot \subseteq \mathsf{w}$ is inhabited.

Hence, from hypothesis (2.10), we can conclude $(\mathsf{f}\ \bot) \subseteq \mathsf{w}$. But then, by applying (2.10) again, $\mathsf{f}\ (\mathsf{f}\ \bot) \subseteq \mathsf{w}$ holds, which means $\mathsf{f}\ (\mathsf{f}\ (\mathsf{f}\ \bot)) \subseteq \mathsf{w}$ holds, and so on.

In other words, hypothesis (2.10) is implicitly claiming that each element in the sequence:

$$\bot, \quad \mathsf{f}\ \bot, \quad \mathsf{f}\ (\mathsf{f}\ \bot), \quad \mathsf{f}\ (\mathsf{f}\ (\mathsf{f}\ \bot)), \quad \ldots$$

is contained in w. Each application of f gets us closer to a fixpoint. Therefore, if we continue this iterative process, w must contain some fixpoint, which implies that it contains the least fixpoint.

The following induction principle specializes the Mendler-style induction principle. It chooses y in the hypothesis so that it is contained not only in w, but also in $\mathsf{Lfp}\ \mathsf{f}$.

Corollary 2.4.4 (Extended Mendler-style induction principle)**.** *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \ldots)$ *be a complete lattice,* $\mathsf{f} : \mathsf{L} \to \mathsf{L}$ *a monotone function, and* $\mathsf{w} : \mathsf{L}$ *an element in the complete lattice. Then, the following mere proposition is inhabited:*

$$\left(\prod_{\mathsf{y}:\mathsf{L}} (\mathsf{y} \leq \mathsf{Lfp}\ \mathsf{f}) \to (\mathsf{y} \leq \mathsf{w}) \to (\mathsf{f}\ \mathsf{y} \leq \mathsf{w})\right) \to (\mathsf{Lfp}\ \mathsf{f} \leq \mathsf{w})$$

Proof. Let $\mathsf{h} : \prod_{\mathsf{y}:\mathsf{L}} (\mathsf{y} \leq \mathsf{Lfp}\ \mathsf{f}) \to (\mathsf{y} \leq \mathsf{w}) \to (\mathsf{f}\ \mathsf{y} \leq \mathsf{w})$.

By Lemma 2.2.11(i), there is a term $\mathsf{t}_1 : (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{Lfp}\ \mathsf{f}$. Also, by Lemma 2.2.11(iii), there is a term $\mathsf{t}_2 : (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{w}$.

Then, $\mathsf{t}_3 :\equiv \mathsf{h}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w})\ \mathsf{t}_1\ \mathsf{t}_2 : \mathsf{f}\ (\mathsf{Lfp}\ \mathsf{f} \wedge \mathsf{w}) \leq \mathsf{w}$. Therefore, term t_3 is a proof that the condition on the extended conventional induction principle holds, which implies that $\mathsf{Lfp}\ \mathsf{f} \leq \mathsf{w}$ is inhabited.

□

Now, the coinduction principles. These principles are duals to all previous induction principles.

The conventional coinduction principle reads: "To prove that all elements in w are in $\mathsf{Gfp}\ \mathsf{f}$, prove that w is contained on its improvement". In other words, *if* either w lacks points (because w falls inside its improvement $\mathsf{f}\ \mathsf{w}$) or w is already a fixpoint, *then* w *must* be contained in some fixpoint, and as a consequence, it will be contained in the greatest fixpoint.

Corollary 2.4.5 (Conventional coinduction principle)**.** *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \ldots)$ *be a complete lattice,* $\mathsf{f} : \mathsf{L} \to \mathsf{L}$ *a monotone function, and* $\mathsf{w} : \mathsf{L}$ *an element in the complete lattice. Then, the following mere proposition is inhabited:*

$$(\mathsf{w} \leq \mathsf{f}\ \mathsf{w}) \to (\mathsf{w} \leq \mathsf{Gfp}\ \mathsf{f})$$

Proof. It follows by duality. Just apply the conventional induction principle on the dual lattice A^{-1}, as constructed in Example 2.2.7.

□

The next coinduction principle states that it is enough to check that w is contained in the improvement of the union of w and $\mathsf{Gfp}\ \mathsf{f}$. It is dual to the extended induction principle.

Corollary 2.4.6 (Extended conventional coinduction principle)**.** *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \ldots)$ *be a complete lattice,* $\mathsf{f} : \mathsf{L} \to \mathsf{L}$ *a monotone function, and* $\mathsf{w} : \mathsf{L}$ *an element in the complete lattice. Then, the following mere proposition is inhabited:*

$$(\mathsf{w} \leq \mathsf{f}\ (\mathsf{Gfp}\ \mathsf{f} \vee \mathsf{w})) \to (\mathsf{w} \leq \mathsf{Gfp}\ \mathsf{f})$$

Proof. It follows by duality. Just apply the extended conventional induction principle on the dual lattice A^{-1}.

□

We explained that Mendler-style induction claims that if each element on the sequence:

$$\bot, \quad \mathsf{f}\ \bot, \quad \mathsf{f}\ (\mathsf{f}\ \bot), \quad \mathsf{f}\ (\mathsf{f}\ (\mathsf{f}\ \bot)), \quad \ldots$$

is contained in w (where $\bot$ is the empty subtype), then w must contain the least fixpoint.

Dually, Mendler-style coinduction claims that if each element on the sequence:

$$\top, \quad \mathsf{f}\ \top, \quad \mathsf{f}\ (\mathsf{f}\ \top), \quad \mathsf{f}\ (\mathsf{f}\ (\mathsf{f}\ \top)), \quad \ldots$$

contains w, where $\top$ is the full subtype $\top :\equiv (\lambda \mathsf{Y}.\ \mathbb{1}) : \mathsf{T} \to \mathbf{Prop}$, then w must be contained in some fixpoint, and as a consequence, w will be contained in the greatest fixpoint.

Corollary 2.4.7 (Mendler-style coinduction principle)**.** *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \dots)$ *be a complete lattice,* $\mathsf{f} : \mathsf{L} \to \mathsf{L}$ *a monotone function, and* $\mathsf{w} : \mathsf{L}$ *an element in the complete lattice. Then, the following mere proposition is inhabited:*

$$\left(\prod_{\mathsf{y}:\mathsf{L}} (\mathsf{w} \leq \mathsf{y}) \to (\mathsf{w} \leq \mathsf{f}\ \mathsf{y})\right) \to (\mathsf{w} \leq \mathsf{Gfp}\ \mathsf{f})$$

Proof. It follows by duality. Just apply the Mendler-style induction principle on the dual lattice A^{-1}.

□

The following coinduction principle specializes Mendler-style coinduction. It chooses y in the hypothesis so that it contains not only w, but also $\mathsf{Gfp}\ \mathsf{f}$.

Corollary 2.4.8 (Extended Mendler-style coinduction principle)**.** *Let* $\mathsf{A} :\equiv (\mathsf{L}, \leq, \dots)$ *be a complete lattice,* $\mathsf{f} : \mathsf{L} \to \mathsf{L}$ *a monotone function, and* $\mathsf{w} : \mathsf{L}$ *an element in the complete lattice. Then, the following mere proposition is inhabited:*

$$\left(\prod_{\mathsf{y}:\mathsf{L}} (\mathsf{Gfp}\ \mathsf{f} \leq \mathsf{y}) \to (\mathsf{w} \leq \mathsf{y}) \to (\mathsf{w} \leq \mathsf{f}\ \mathsf{y})\right) \to (\mathsf{w} \leq \mathsf{Gfp}\ \mathsf{f})$$

Proof. It follows by duality. Just apply the extended Mendler-style induction principle on the dual lattice A^{-1}.

□

2.5 Chapter summary

This chapter developed in HoTT classical results regarding the existence of fixpoints for monotone functions on a complete lattice. This provided an opportunity to show an example on how set-based mathematics can be formalized in HoTT.

Sections 2.1 and 2.2 defined the concepts of poset, monotone function, complete lattice, infima, suprema, meet, and join. The dual and powertype lattices were constructed as examples. Also, the powertype lattice served as an example on how the propositional resizing axiom is used for impredicative definitions. Section 2.3 defined the concepts of prefixpoint, postfixpoint, and least and greatest fixpoints. It also provided a proof for the Knaster-Tarski Lemma. Finally, Section 2.4 constructed induction and coinduction principles as corollaries to the Knaster-Tarski Lemma.

Chapter 3

Constructing types in HoTT

This chapter explores the construction of inductive and coinductive types *inside* HoTT. In other words, it explores the construction of types without adding more inference rules than those already present in Chapter 1.

Sections 1.9 and 1.10 explained that inductive and coinductive types can be modeled by initial algebras and final coalgebras, respectively. Also, these sections mentioned that an arbitrary functor does not necessarily have an initial algebra and a final coalgebra.

This chapter explores the initial/final (co)algebra existence question for an endofunctor.[1] The standard approach found in the literature involves the use of chains and cochains under suitable conditions on the functor (see introduction in the next section). But, instead of reproducing results involving chains and cochains, which can be found formalized in HoTT (see for example [4]), we will attempt to *translate* the Knaster-Tarski construction done in the proof for Lemma 2.3.6 into the language of algebras and coalgebras. Can we get initial algebras and final coalgebras by doing such thing? If not, how close are the constructed (co)algebras from being initial or final? Are these constructions useful at all? This chapter explores some implications on performing such experiment.

One consequence of the proposed experiment is that we only require very mild conditions on the involved endofunctor. For example, it needs to preserve sets, but there is no need to require that the functor preserves colimits of chains and limits of cochains, as it is required in standard approaches (see introduction in the next section).

However, this generality comes with a cost. The (co)algebras fail to be initial or final only by a universe level. In other words, they behave like initial algebras and final coalgebras, except for a restriction on universe levels.[2] We will call these constructions *lower level initial algebras* and *lower level final coalgebras* for reasons we will make precise later. Another limitation on the approach is that we can only build sets instead of general types.

Although the proposed experiment does not answer positively the initial/final (co)algebra existence question, we will see that the lower level initial/final (co)algebras constructed by the experiment are still useful, as they are able to define *unique* functions by (co)recursive equations.

Another important point to be highlighted on this chapter is the explicit use of universe levels. In contrast to Chapter 2 where the theory was developed by ignoring universe levels, we will see that the results on this chapter *depend critically* on the explicit use of universe levels. The results developed on this chapter are among those rare occasions where ignoring universe levels leads to an apparent inconsistency in the type theory, justifying their explicit use. If Chapter 2 was an example for the "ignore universe levels" style of doing mathematics in HoTT, this chapter will serve as an example for the "explicit universe levels" style.

This chapter is structured as follows:

- Section 3.1 presents an overview of the initial/final (co)algebra existence problem. It describes the most common approach found in the literature, which involves the computation of colimits of ω-chains and limits of ω-cochains.
- Sections 3.2 and 3.3 develop the suggested experiment for inductive and coinductive types, respectively.

  These sections prove the existence theorems for lower level initial/final (co)algebras. Also, they show that the "powertype functor" has a lower level initial/final (co)algebra, but not an initial/final (co)algebra.

[1] See Definition 1.9.8.

[2] This "restriction" will become clear as the results are developed and some examples are presented.

These sections also provide examples on how lower level initial/final (co)algebras are used: the natural numbers seen as a lower level initial algebra, and the streams seen as a lower level final coalgebra.

Finally, each section discusses the limitations on the approach, and explains why certain decisions were made by the author.

The results on this chapter have been computer-checked in the Coq proof assistant [24]. See [26] for details on how to download the proof script file for this chapter.

3.1 Introduction

For this section, we will work in standard mathematics. The author assumes that the reader is already familiar with basic concepts like partial order, monotone function, complete lattice, least upper bound (supremum), and greatest lower bound (infimum), as presented in standard mathematics (see [9] and [16] for references).

The procedure commonly used for constructing initial and final (co)algebras is a generalization of an order-theoretic result for constructing least and greatest fixpoints of monotone functions on a complete lattice. This order-theoretic result differs from the Knaster-Tarski Lemma (see Lemma 2.3.6) in the sense that it computes fixpoints by explicit iteration. We explain this alternative order-theoretic result before explaining its generalization into the language of algebras and coalgebras.

The "iterative procedure" will require that monotone functions satisfy an extra condition, which we define next (see Definition 2.8.1 in [16]).

Definition 3.1.1 (Continuous and cocontinuous monotone function)**.** Let $\langle L, \leq, \bigwedge, \bigvee \rangle$ be a complete lattice[3] and $g : L \to L$ a monotone function. We say that:

- Function g is *continuous* if every increasing sequence in L:

$$\alpha_0 \;\leq\; \alpha_1 \;\leq\; \alpha_2 \;\leq\; \dots$$

  satisfies $g(\bigvee\{\alpha_i \mid i \in \mathbb{N}\}) = \bigvee\{g(\alpha_i) \mid i \in \mathbb{N}\}$, i.e. if g preserves the supremum of any increasing sequence.

- Function g is *cocontinuous* if every decreasing sequence in L:

$$\alpha_0 \;\geq\; \alpha_1 \;\geq\; \alpha_2 \;\geq\; \dots$$

  satisfies $g(\bigwedge\{\alpha_i \mid i \in \mathbb{N}\}) = \bigwedge\{g(\alpha_i) \mid i \in \mathbb{N}\}$, i.e. if g preserves the infimum of any decreasing sequence.

▲

Every complete lattice has a bottom element $\bot :\equiv \bigvee \emptyset$ and a top element $\top :\equiv \bigwedge \emptyset$. Hence, for every monotone function $g : L \to L$, we can obtain an increasing sequence (since $\bot$ is a bottom element):

$$\bot \;\leq\; g(\bot) \;\leq\; g(g(\bot)) \;\leq\; g(g(g(\bot))) \;\leq\; \dots$$

and a decreasing sequence (since $\top$ is a top element):

$$\top \;\geq\; g(\top) \;\geq\; g(g(\top)) \;\geq\; g(g(g(\top))) \;\geq\; \dots$$

We can rewrite these sequences as:

$$\begin{aligned} g^0(\bot) \;\leq\; g^1(\bot) \;\leq\; g^2(\bot) \;\leq\; g^3(\bot) \;\leq\; \dots \\ g^0(\top) \;\geq\; g^1(\top) \;\geq\; g^2(\top) \;\geq\; g^3(\top) \;\geq\; \dots \end{aligned} \tag{3.1}$$

where g^n denotes the n-times composition of g (here, g^0 is defined to be the identity function).

The "iterative procedure" is then encoded in the following result (see Theorem 2.8.5 in [16]).

[3] $\leq$ denotes the partial order relation, $\bigwedge$ the infimum operation, and $\bigvee$ the supremum operation.

Theorem 3.1.2. *Let $\langle L, \leq, \bigwedge, \bigvee \rangle$ be a complete lattice and $g : L \to L$ a monotone function.*

(i) If g is continuous, then $\bigvee\{g^i(\bot) \mid i \in \mathbb{N}\}$ is the least fixpoint of g, i.e. if we iterate g starting from the bottom element (producing an increasing sequence), the least fixpoint of g will be the supremum of the sequence.

(ii) If g is cocontinuous, then $\bigwedge\{g^i(\top) \mid i \in \mathbb{N}\}$ is the greatest fixpoint of g, i.e. if we iterate g starting from the top element (producing a decreasing sequence), the greatest fixpoint of g will be the infimum of the sequence.

Now, the common approach for building initial and final (co)algebras is just a generalization of Theorem 3.1.2, but expressed in the language of category theory, which we explain next.

A category (Definition 1.9.1) will serve as a generalization to a partially ordered set. An arrow $f : A \to B$ in a category serves as a generalization to the statement "$A \leq B$" in a partially ordered set.

Next, we define a generalization to the concept of increasing and decreasing sequences (see Definition 2.9 in [8]).

Definition 3.1.3 (ω-chain and ω-cochain)**.** Let $\mathcal{C}$ be a category. We define:

- An *ω-chain* in $\mathcal{C}$ is a collection $\{O_i\}_{i\in\mathbb{N}}$ of objects in $\mathcal{C}$ together with a collection of arrows $\{\alpha_i : O_i \to O_{i+1}\}_{i\in\mathbb{N}}$.
$$O_0 \xrightarrow{\alpha_0} O_1 \xrightarrow{\alpha_1} O_2 \xrightarrow{\alpha_2} O_3 \xrightarrow{\alpha_3} \dots$$
- An *ω-cochain* in $\mathcal{C}$ is a collection $\{O_i\}_{i\in\mathbb{N}}$ of objects in $\mathcal{C}$ together with a collection of arrows $\{\alpha_i : O_{i+1} \to O_i\}_{i\in\mathbb{N}}$.
$$O_0 \xleftarrow{\alpha_0} O_1 \xleftarrow{\alpha_1} O_2 \xleftarrow{\alpha_2} O_3 \xleftarrow{\alpha_3} \dots$$

▲

An ω-chain generalizes the idea of an increasing sequence in a poset, because its diagram translates to:

$$O_0 \leq O_1 \leq O_2 \leq O_3 \leq \dots$$

Similarly, an ω-cochain generalizes the idea of a decreasing sequence.

An upper bound for an increasing sequence and a lower bound for a decreasing sequence can be generalized through the concepts of cocone for an ω-chain and cone for an ω-cochain, respectively (see Definition 2.11 in [8]).

Definition 3.1.4 (Cocone for ω-chain and cone for ω-cochain)**.** Let $\mathcal{C}$ be category. We define:

- A *cocone* for the ω-chain $(\{O_i\}, \{\alpha_i : O_i \to O_{i+1}\})$ is an object C together with a collection of arrows $\{\beta_i : O_i \to C\}_{i\in\mathbb{N}}$ such that for every $i \in \mathbb{N}$, we have $\beta_i = \beta_{i+1} \circ \alpha_i$.

  In other words, the following diagram commutes for every $i \in \mathbb{N}$:
$$\begin{array}{ccccccc} \dots & \xrightarrow{\alpha_{i-1}} & O_i & \xrightarrow{\alpha_i} & O_{i+1} & \xrightarrow{\alpha_{i+1}} & \dots \\ & & {\scriptstyle\beta_i}\downarrow & \swarrow{\scriptstyle\beta_{i+1}} & & & \\ & & C & & & & \end{array}$$

  The collection $\{\beta_i : O_i \to C\}_{i\in\mathbb{N}}$ expresses that C is "above" all objects O_i. Hence, C is an "upper bound" for the "increasing sequence".

- A *cone* for the ω-cochain $(\{O_i\}, \{\alpha_i : O_{i+1} \to O_i\})$ is an object C together with a collection of arrows $\{\beta_i : C \to O_i\}_{i\in\mathbb{N}}$ such that for every $i \in \mathbb{N}$, we have $\beta_i = \alpha_i \circ \beta_{i+1}$.

  In other words, the following diagram commutes for every $i \in \mathbb{N}$:
$$\begin{array}{ccccccc} & & C & & & & \\ & & {\scriptstyle\beta_i}\downarrow & \searrow{\scriptstyle\beta_{i+1}} & & & \\ \dots & \xleftarrow{\alpha_{i-1}} & O_i & \xleftarrow{\alpha_i} & O_{i+1} & \xleftarrow{\alpha_{i+1}} & \dots \end{array}$$

  The collection $\{\beta_i : C \to O_i\}_{i\in\mathbb{N}}$ expresses that C is "below" all objects O_i. Hence, C is a "lower bound" for the "decreasing sequence".

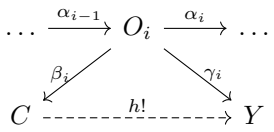
▲

The supremum for an increasing sequence and the infimum for a decreasing sequence follow a similar generalization through the concepts of colimit for an ω-chain and limit for an ω-cochain, respectively (see Definition 2.13 in [8]).

Definition 3.1.5 (Colimit for ω-chain and limit for ω-cochain)**.** Let $\mathcal{C}$ be a category. We define:

- A *colimit* for the ω-chain $(\{O_i\}, \{\alpha_i : O_i \to O_{i+1}\})$ is a cocone $(C, \{\beta_i : O_i \to C\})$ such that for any other cocone $(Y, \{\gamma_i : O_i \to Y\})$, there is a *unique* arrow $h : C \to Y$ such that for any $i \in \mathbb{N}$, we have $\gamma_i = h \circ \beta_i$.

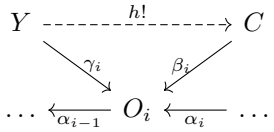
  In other words, the following diagram commutes for every $i \in \mathbb{N}$:



  The collection $\{\gamma_i : O_i \to Y\}$ means that Y is an "upper bound" for the "increasing sequence" (similarly for C), and arrow $h : C \to Y$ means that C is "smaller" than Y. Hence, C is the "least upper bound" (as Y is arbitrary).

- A *limit* for the ω-cochain $(\{O_i\}, \{\alpha_i : O_{i+1} \to O_i\})$ is a cone $(C, \{\beta_i : C \to O_i\})$ such that for any other cone $(Y, \{\gamma_i : Y \to O_i\})$, there is a *unique* arrow $h : Y \to C$ such that for any $i \in \mathbb{N}$, we have $\gamma_i = \beta_i \circ h$.

  In other words, the following diagram commutes for every $i \in \mathbb{N}$:



  The collection $\{\gamma_i : Y \to O_i\}$ means that Y is a "lower bound" for the "decreasing sequence" (similarly for C), and arrow $h : Y \to C$ means that C is "bigger" than Y. Hence, C is the "greatest lower bound" (as Y is arbitrary).

▲

Notice that the statement for Theorem 3.1.2 depends only on our ability to obtain the supremum of an increasing sequence and the infimum of a decreasing sequence. Therefore, we do not need a complete lattice, but only a poset where every increasing sequence has a supremum and every decreasing sequence has an infimum. Also, our poset needs to have a top element and a bottom element.

These conditions can be expressed in category theory by claiming that category $\mathcal{C}$ (i.e. our poset) needs to have colimits for any ω-chain (i.e. any increasing sequence has a supremum) and limits for any ω-cochain (i.e. any decreasing sequence has an infimum). Also, $\mathcal{C}$ needs to have an initial object 0 (i.e. a bottom element), and a final object 1 (i.e. a top element). Therefore, we make the following definitions (see Definition 2.15 in [8]).

Definition 3.1.6 (ω-complete and ω-cocomplete category)**.** Let $\mathcal{C}$ be a category. We say that:

- Category $\mathcal{C}$ is ω-*complete* if it has an initial object, and every ω-chain has a colimit.
- Category $\mathcal{C}$ is ω-*cocomplete* if it has a terminal object, and every ω-cochain has a limit.

▲

An example of a category that is ω-complete and ω-cocomplete is `SET` (the category of sets and functions).[4]

Now, a functor (Definition 1.9.2) acts as a generalization to the concept of monotone function, i.e. just notice that the arrow mapping $(A \to B) \to (H(A) \to H(B))$ can be interpreted as the monotonicity statement "$(A \leq B) \to (H(A) \leq H(B))$".

The concepts in Definition 3.1.1 can be generalized to a functor as follows.

[4] It follows by Corollary 5.22 and Theorem 5.23 in [5].

Definition 3.1.7 (Preservation of (co)limits of ω-(co)chains)**.** Let $\mathcal{C}$ be a category and $H : \mathcal{C} \to \mathcal{C}$ a functor.

- We say that functor H *preserves colimits of ω-chains* if given a colimit $(C, \{\beta_i\})$ for the ω-chain $(\{O_i\}, \{\alpha_i\})$, we have that $(H(C), \{H(\beta_i)\})$ is also a colimit for the ω-chain $(\{H(O_i)\}, \{H(\alpha_i)\})$.

  Stating that $(C, \{\beta_i\})$ is a colimit for $(\{O_i\}, \{\alpha_i\})$ is analogous to expressing "$C = \bigvee\{O_i\}$" in the language of order theory (since colimits are unique up to isomorphism). Similarly, stating that $(H(C), \{H(\beta_i)\})$ is a colimit for $(\{H(O_i)\}, \{H(\alpha_i)\})$ is analogous to expressing "$H(C) = \bigvee\{H(O_i)\}$" in the language of order theory.

  Therefore, it is as if we are expressing "$H(\bigvee\{O_i\}) = \bigvee\{H(O_i)\}$". In other words, our functor is "continuous" because it preserves the "supremum of increasing sequences".

- We say that functor H *preserves limits of ω-cochains* if given a limit $(C, \{\beta_i\})$ for the ω-cochain $(\{O_i\}, \{\alpha_i\})$, we have that $(H(C), \{H(\beta_i)\})$ is also a limit for the ω-cochain $(\{H(O_i)\}, \{H(\alpha_i)\})$.

  Again, a similar analysis shows that it is as if we are expressing "$H(\bigwedge\{O_i\}) = \bigwedge\{H(O_i)\}$" in the language of order theory. In other words, our functor is "cocontinuous" because it preserves the "infimum of decreasing sequences".

▲

We also have generalizations for the increasing and decreasing sequences in (3.1).

Definition 3.1.8 (Initial ω-chain and final ω-cochain)**.** Let $\mathcal{C}$ be a category and $H : \mathcal{C} \to \mathcal{C}$ a functor.

- If $\mathcal{C}$ has an initial object 0, then we define the *initial ω-chain*:

$$0 \xrightarrow{!} H^1(0) \xrightarrow{H^1(!)} H^2(0) \xrightarrow{H^2(!)} H^3(0) \xrightarrow{H^3(!)} \dots$$

  where ! is the unique arrow going out of initial object 0, arrow $H^n(!)$ denotes the n-application of functor H on arrow !, and $H^n(0)$ denotes the n-application of functor H on object 0.

  The initial ω-chain is analogous to the increasing sequence:

$$g^0(\bot) \leq g^1(\bot) \leq g^2(\bot) \leq g^3(\bot) \leq \dots$$

  for a monotone function g.

- If $\mathcal{C}$ has a final object 1, then we define the *final ω-cochain*:

$$1 \xleftarrow{!} H^1(1) \xleftarrow{H^1(!)} H^2(1) \xleftarrow{H^2(!)} H^3(1) \xleftarrow{H^3(!)} \dots$$

  where ! is the unique arrow going into terminal object 1, arrow $H^n(!)$ denotes the n-application of functor H on arrow !, and $H^n(1)$ denotes the n-application of functor H on object 1.

  The final ω-cochain is analogous to the decreasing sequence:

$$g^0(\top) \geq g^1(\top) \geq g^2(\top) \geq g^3(\top) \geq \dots$$

  for a monotone function g.

▲

Next, we need to define generalizations for the concepts of prefixpoint, postfixpoint and fixpoint found in order theory (see Definition 2.3.1).

An H-algebra for a functor $H : \mathcal{C} \to \mathcal{C}$ (Definition 1.9.3) was defined as a pair (A, In_A), where A is an object and $In_A : H(A) \to A$ is an arrow. But the arrow $H(A) \to A$ reads as the statement "$H(A) \leq A$". Hence, an H-algebra can be interpreted as a "prefixpoint" for "monotone function" H.

Similarly, an H-coalgebra for a functor $H : \mathcal{C} \to \mathcal{C}$ (Definition 1.10.1) was defined as a pair (A, Out_A), where A is an object and $Out_A : A \to H(A)$ is an arrow. But the arrow $A \to H(A)$ reads as the statement "$A \leq H(A)$". Hence, an H-coalgebra can be interpreted as a "postfixpoint" for "monotone function" H.

Now, (A, In_A) is an initial H-algebra (Definition 1.9.5) if for any other H-algebra (B, In_B), there is a unique algebra morphism $A \to B$. But the arrow $A \to B$ reads as the statement "$A \leq B$". Hence, an initial algebra can be interpreted as a "prefixpoint" A that is "smaller" than any other "prefixpoint" B, i.e. the "smallest prefixpoint" for "monotone function" H.

By a similar argument, a final H-coalgebra (Definition 1.10.3) can be interpreted as the "greatest postfixpoint" for "monotone function" H.

However, if A is the smallest prefixpoint for a monotone function, then it is actually the smallest fixpoint. Similarly, if A is the greatest postfixpoint for a monotone function, then it is actually the greatest fixpoint. This is supported by two results due to Lambek (see Lemma 2.5.5 and Lemma 2.6.4 in [17]), which state that if A is an initial H-algebra or a final H-coalgebra, then A and $H(A)$ are isomorphic objects.

So, an initial H-algebra is a generalization to the concept of least fixpoint for a monotone function, and a final H-coalgebra is a generalization to the concept of greatest fixpoint for a monotone function.

With all these concepts at hand, we are ready to state a generalized version of Theorem 3.1.2 (see Theorems 2.1.9 and 2.3.3 in [1]).

Theorem 3.1.9. *Let $\mathcal{C}$ be a category and $H : \mathcal{C} \to \mathcal{C}$ a functor.*

(i) Suppose $\mathcal{C}$ is ω-complete and functor H preserves colimits of ω-chains. Let C be the colimit of the initial ω-chain. Then, there is an arrow $In_C : H(C) \to C$ such that (C, In_C) is an initial H-algebra.

(ii) Suppose $\mathcal{C}$ is ω-cocomplete and functor H preserves limits of ω-cochains. Let C be the limit of the final ω-cochain. Then, there is an arrow $Out_C : C \to H(C)$ such that (C, Out_C) is a final H-coalgebra.

Although it is possible to formalize all these results in HoTT,[5] this report will take another route.

What could happen if we attempt a direct generalization of the Knaster-Tarski construction (Lemma 2.3.6) using the analogies presented on this section? Can we get initial and final (co)algebras by doing such thing? If not, can we obtain something useful out of it? Will the results be general, in the sense that they apply to an arbitrary endofunctor (in comparison to Theorem 3.1.9, where the endofunctor needs to preserve (co)limits of ω-(co)chains)?

The following sections flesh out the details of such experiment, first for inductive types or initial algebras (Section 3.2), and then for coinductive types or final coalgebras (Section 3.3).

3.2 Inductive types

This section describes the construction of inductive types by means of generalizing the Knaster-Tarski construction (Lemma 2.3.6(i)) into the language of algebras.

We will see that generalizing Knaster-Tarski produces lower level *weakly* initial algebras (this concept will be made precise later). Therefore, to obtain a (lower level) initial algebra from a (lower level) weakly initial algebra, we need to devise a method of "refinement".

Hence, the development will be split into two parts. Subsection 3.2.1 carries out the generalization of Knaster-Tarski, and Subsection 3.2.2 carries out the "refinement" of (lower level) weakly initial algebras into (lower level) initial algebras.

3.2.1 Generalizing the Knaster-Tarski construction

As was discussed in Section 1.9, we will not be interested on building initial/final (co)algebras on general categories, but only on type universes. All concepts discussed in Section 3.1 apply to type universes, because they can be seen as categories (see discussion on Section 1.9). Hence, our functors will be maps between type universes (see Definition 1.9.6).

As suggested on Section 3.1, a functor is a generalization to the concept of monotone function (Definition 2.1.6), the arrow type $\to$ acts as a generalization to the order relation $\leq$ (i.e. $\mathsf{A} \to \mathsf{B}$ corresponds to "$\mathsf{A} \leq \mathsf{B}$"), and an H-algebra acts as a generalization to the concept of prefixpoint of a monotone function.

Now, if we examine the proof for the first part of Knaster-Tarski (Lemma 2.3.6(i)), we see that the least fixpoint was defined as the infimum of all prefixpoints of the monotone function. Therefore, the only missing ingredient is a generalization to the concept of "greatest lower bound of a subset in the lattice".

[5] See [4] for an example on how these ideas can be formalized in HoTT. In [4], they prove the existence of final coalgebras for "polynomial" functors, using some of the ideas presented on this section.

Again, as suggested on Section 3.1, a category (with additional conditions) generalizes the concept of lattice. In our case, type universes play the role of "categories", hence type universes will be our "lattices".

Therefore, a "subset of elements in the lattice" gets translated to a "collection of types in a type universe". To represent collections of types in a type universe $\mathcal{U}_j$, we will use type families $\mathsf{M} \to \mathcal{U}_j$, where M is some indexing type. In other words, a function $\mathsf{F} : \mathsf{M} \to \mathcal{U}_j$ will be *analogous* to a "family of collections" $\{F_i\}_{i \in M}$ indexed by collection M, as understood in standard mathematics.

With this in mind, a "lower bound" for a family $\mathsf{F} : \mathsf{M} \to \mathcal{U}_j$ will be a type $\mathsf{A} : \mathcal{U}_k$ that is "below" every element in the family (i.e. for every $\mathsf{i} : \mathsf{M}$, we have a proof for $\mathsf{A} \to (\mathsf{F}\ \mathsf{i})$, or interpreted order-theoretically, a proof for "$\mathsf{A} \leq \mathsf{F}\ \mathsf{i}$"). By borrowing terminology from category theory, this generalized notion of "lower bound" corresponds to the notion of *cone* over the family F.

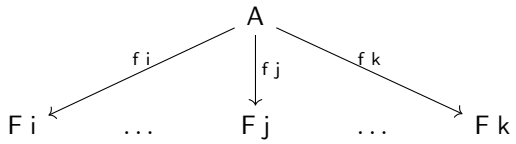
Definition 3.2.1 (Cone over a family)**.** Let $\mathsf{M} : \mathcal{U}_i$ be an indexing type, and $\mathsf{F} : \mathsf{M} \to \mathcal{U}_j$ a family of types. Let k be a universe level. Any term of type:

$$\mathsf{Cone}_k\ \mathsf{F} :\equiv \sum_{\mathsf{A}:\mathcal{U}_k} \prod_{\mathsf{i}:\mathsf{M}} \mathsf{A} \to (\mathsf{F}\ \mathsf{i})$$

will be called a *cone* over the family F at level k.

▲

Intuitively, $\mathsf{Cone}_k\ \mathsf{F}$ is the type of all "lower bounds" for the family F. Observe that a cone over F is a type A *together* with a family of functions $\mathsf{f} : \prod_{\mathsf{i}:\mathsf{M}} \mathsf{A} \to (\mathsf{F}\ \mathsf{i})$. In other words, we have the diagram:



Now, the "greatest lower bound" will be the biggest among the lower bounds. If we try to generalize Definition 2.2.3(ii), we will get the following:

$$\left(\prod_{\mathsf{i}:\mathsf{M}} \mathsf{A} \to (\mathsf{F}\ \mathsf{i})\right) \times \left(\prod_{(\mathsf{Y},\mathsf{g}):\mathsf{Cone}_k\ \mathsf{F}} \mathsf{Y} \to \mathsf{A}\right)$$

Therefore, the type of all generalized greatest lower bounds for F will look like:

$$\sum_{\mathsf{A}:\mathcal{U}_k} \left[\left(\prod_{\mathsf{i}:\mathsf{M}} \mathsf{A} \to (\mathsf{F}\ \mathsf{i})\right) \times \left(\prod_{(\mathsf{Y},\mathsf{g}):\mathsf{Cone}_k\ \mathsf{F}} \mathsf{Y} \to \mathsf{A}\right)\right]$$

which can be rewritten as:

$$\sum_{(\mathsf{A},\mathsf{f}):\mathsf{Cone}_k\ \mathsf{F}} \prod_{(\mathsf{Y},\mathsf{g}):\mathsf{Cone}_k\ \mathsf{F}} \mathsf{Y} \to \mathsf{A} \tag{3.2}$$

by associativity of sigma types (i.e. expand definitions and apply Lemma 1.6.23).

In other words, A will be a "greatest lower bound" for F if A is a "lower bound" for F (i.e. it forms a cone for F), and any other "lower bound" Y (i.e. a cone over F) is "below" A. However, there may be many different proofs of $\mathsf{Y} \to \mathsf{A}$ for every $(\mathsf{Y}, \mathsf{g}) : \mathsf{Cone}_k\ \mathsf{F}$, since, in this case, the arrow type is not a mere proposition. This is a completely different situation to what happened in Definition 2.2.3(ii), where the order relation $\leq$ was a mere proposition (and hence, proofs were irrelevant).

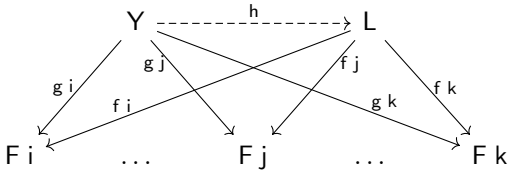
So, this confronts us with the problem of which function should we choose as proof of $\mathsf{Y} \to \mathsf{A}$ for each $(\mathsf{Y}, \mathsf{g}) : \mathsf{Cone}_k\ \mathsf{F}$. It turns out that a "useful" condition the chosen function should satisfy is captured by the notion of *weak limit* (borrowing again from category theory). Weak limits will provide us with a working generalization to the concept of "greatest lower bound".[6]

[6] Although we may use the stronger notion of *limit* (see [5]), the notion of weak limit will be enough for the purposes on this report, see Remark 3.2.8.

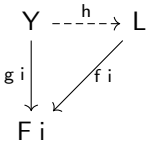
Definition 3.2.2 (Weak limit of a type family)**.** Let $\mathsf{M}:\mathcal{U}_i$ be an indexing type, and $\mathsf{F}:\mathsf{M}\to\mathcal{U}_j$ a family of types. Let k be a universe level. Any term of type:

$$\mathsf{WLimit}_k\ \mathsf{F} :\equiv \sum_{(\mathsf{L},\mathsf{f}):\mathsf{Cone}_k\ \mathsf{F}}\ \prod_{(\mathsf{Y},\mathsf{g}):\mathsf{Cone}_k\ \mathsf{F}}\ \sum_{\mathsf{h}:\mathsf{Y}\to\mathsf{L}}\ \prod_{\mathsf{i}:\mathsf{M}}\ \prod_{\mathsf{w}:\mathsf{Y}} \mathsf{g\ i\ w} = \mathsf{f\ i\ (h\ w)}$$

will be called a *weak limit* for the family F at level k. In other words, we have the commutative diagram:



i.e. the cone (L,f) will be a weak limit for F if for any other cone (Y,g), there is a function $\mathsf{h}:\mathsf{Y}\to\mathsf{L}$ such that for every $\mathsf{i}:\mathsf{M}$, the following triangle commutes:



▲

Notice that type $\mathsf{WLimit}_k\ \mathsf{F}$ is just (3.2) augmented with the "commutativity condition":

$$\prod_{\mathsf{i}:\mathsf{M}}\ \prod_{\mathsf{w}:\mathsf{Y}} \mathsf{g\ i\ w} = \mathsf{f\ i\ (h\ w)}$$

With this "commutativity condition", we are effectively claiming that functions $\mathsf{f}:\prod_{\mathsf{i}:\mathsf{M}}\mathsf{A}\to(\mathsf{F\ i})$, $\mathsf{g}:\prod_{\mathsf{i}:\mathsf{M}}\mathsf{Y}\to(\mathsf{F\ i})$, and $\mathsf{h}:\mathsf{Y}\to\mathsf{A}$ in type (3.2) are not chosen arbitrarily (which would be implied if we were to use type (3.2) instead of $\mathsf{WLimit}_k\ \mathsf{F}$).

The first result states that any family has a weak limit (i.e. any family has a "greatest lower bound").

Lemma 3.2.3. *Let $\mathsf{M}:\mathcal{U}_i$ be an indexing type, and $\mathsf{F}:\mathsf{M}\to\mathcal{U}_j$ a family of types. Then, the following type is inhabited:*

$$\mathsf{WLimit}_{\max\{i,j\}}\ \mathsf{F}$$

Proof. First, we need to build a tuple $(\mathsf{L},\mathsf{f}):\mathsf{Cone}_{\max\{i,j\}}\ \mathsf{F}$.

In standard mathematics, the greatest lower bound for a family of sets $\{F_i\}_{i\in M}$ is the set $\bigcap_{i\in M} F_i$.

In our case, type $\prod_{\mathsf{i}:\mathsf{M}}\mathsf{F\ i}$ will be analogous to $\bigcap_{i\in M} F_i$.

Define:

$$\mathsf{L} :\equiv \prod_{\mathsf{i}:\mathsf{M}} \mathsf{F\ i}$$

Notice $\mathsf{L}:\mathcal{U}_{\max\{i,j\}}$ since $\mathsf{M}:\mathcal{U}_i$ and $\mathsf{F\ i}:\mathcal{U}_j$ for every $\mathsf{i}:\mathsf{M}$ (see Remark 1.5.16).

Next, we need to build a function $\mathsf{f}:\prod_{\mathsf{i}:\mathsf{M}}\mathsf{L}\to(\mathsf{F\ i})$. Define:

$$\mathsf{f} :\equiv \lambda(\mathsf{i}:\mathsf{M})(\mathsf{j}:\mathsf{L}).\ \mathsf{j\ i}$$

which is well-typed, since $\mathsf{j}:\prod_{\mathsf{i}:\mathsf{M}}\mathsf{F\ i}$.

Now, we need to prove that the following type is inhabited:

$$\prod_{(\mathsf{Y},\mathsf{g}):\mathsf{Cone}_{\max\{i,j\}}\ \mathsf{F}}\ \sum_{\mathsf{h}:\mathsf{Y}\to\mathsf{L}}\ \prod_{\mathsf{i}:\mathsf{M}}\ \prod_{\mathsf{w}:\mathsf{Y}} \mathsf{g\ i\ w} = \mathsf{f\ i\ (h\ w)}$$

Let $(\mathsf{Y},\mathsf{g}):\mathsf{Cone}_{\max\{i,j\}}\ \mathsf{F}$. Define:

$$\mathsf{h} :\equiv (\lambda\mathsf{y}:\mathsf{Y}.\ \lambda\mathsf{i}:\mathsf{M}.\ \mathsf{g\ i\ y}):\mathsf{Y}\to\left(\prod_{\mathsf{i}:\mathsf{M}}\mathsf{F\ i}\right)$$

Now, let $\mathsf{i} : \mathsf{M}$ and $\mathsf{w} : \mathsf{Y}$. Then, we have by definition:

$$\begin{aligned} \mathsf{f}\ \mathsf{i}\ (\mathsf{h}\ \mathsf{w}) &\equiv \mathsf{f}\ \mathsf{i}\ (\lambda \mathsf{i} : \mathsf{M}.\ \mathsf{g}\ \mathsf{i}\ \mathsf{w}) \\ &\equiv (\lambda \mathsf{i} : \mathsf{M}.\ \mathsf{g}\ \mathsf{i}\ \mathsf{w})\ \mathsf{i} \\ &\equiv \mathsf{g}\ \mathsf{i}\ \mathsf{w} \end{aligned}$$

And so, $\mathsf{g}\ \mathsf{i}\ \mathsf{w} = \mathsf{f}\ \mathsf{i}\ (\mathsf{h}\ \mathsf{w})$ is inhabited.

□

Next, we need to define the concept of weakly initial set algebra, but first we need to specialize the concept of H-algebra (Definition 1.9.11) to sets. The reason of why we require to restrict Definition 1.9.11 to sets will be discussed in Section 3.2.5.

Definition 3.2.4 (Set H-algebra)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, and $k \leq i$ a universe level. Any term of type:

$$\mathsf{SAlg}_k\ \mathsf{H} :\equiv \sum_{\mathsf{A}:\mathbf{Set}_k} (\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$$

will be called a *set* H*-algebra* at level k.

▲

The notion of algebra morphism AlgMor A B (Definition 1.9.12) applies to set H-algebras, because any set H-algebra *is* an H-algebra by ignoring that it is a set.

Now, we are ready to define the notion of weakly initial set H-algebra. We will define two versions for it.

The difference between the two versions is in the universe level of the set H-algebra quantified on the Π-type (see Definition 3.2.5 below). In type $\mathsf{LLWinSAlg}_k\ \mathsf{H}$, the level is $k-1$. In type $\mathsf{WinSAlg}_k\ \mathsf{H}$, the level is k.

Definition 3.2.5 ((Lower level) weakly initial set H-algebra)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor.

(i) Let $0 < k \leq i$ be a universe level. Any term of type:

$$\mathsf{LLWinSAlg}_k\ \mathsf{H} :\equiv \sum_{(\mathsf{W},\mathsf{In}_\mathsf{W}):\mathsf{SAlg}_k\ \mathsf{H}} \prod_{(\mathsf{B},\mathsf{In}_\mathsf{B}):\mathsf{SAlg}_{k-1}\ \mathsf{H}} \mathsf{AlgMor}\ \mathsf{W}\ \mathsf{B}$$

will be called a *lower level weakly initial set* H*-algebra* (*LLWinSAlg* for short) at level k.

(ii) Let $k \leq i$ be a universe level. Any term of type:

$$\mathsf{WinSAlg}_k\ \mathsf{H} :\equiv \sum_{(\mathsf{W},\mathsf{In}_\mathsf{W}):\mathsf{SAlg}_k\ \mathsf{H}} \prod_{(\mathsf{B},\mathsf{In}_\mathsf{B}):\mathsf{SAlg}_k\ \mathsf{H}} \mathsf{AlgMor}\ \mathsf{W}\ \mathsf{B}$$

will be called a *weakly initial set* H*-algebra* (*WinSAlg* for short) at level k.

▲

Convention 3.2.6. We will write lower level weakly initial set H-algebras as $(\mathsf{W}, \mathsf{In}_\mathsf{W}, \mathsf{m})$, instead of $((\mathsf{W}, \mathsf{In}_\mathsf{W}), \mathsf{m})$. The same applies to weakly initial set H-algebras.

▲

Both versions are similar to Definition 1.9.15, except that type AlgMor W B is not asserted to be contractible. This means that in a (lower level) weakly initial set algebra, the algebra morphism is not required to be unique.

Also, a LLWinSAlg residing at level k will be weakly initial *relative to* set H-algebras at level $k-1$ (hence the words *lower level* in their name). Instead, a WinSAlg residing at level k will be weakly initial relative to set H-algebras at level k.

The restriction to level $k-1$ in type $\mathsf{LLWinSAlg}_k\ \mathsf{H}$ is the "restriction on universe levels" that was mentioned at the start of this chapter. Lemma 3.2.7 will show that generalizing the Knaster-Tarski construction only produces a LLWinSAlg. This restriction will carry on to the next section, where we will *refine* the LLWinSAlg into a lower level initial set H-algebra. To the contrary, the existence of a WinSAlg for an arbitrary endofunctor can be answered in the negative (see Corollary 3.2.26 for an example of functor without a WinSAlg).

Nevertheless, we state both versions because the refinement process of Section 3.2.2 applies to both a LLWinSAlg and a WinSAlg (i.e. if we start with a LLWinSAlg we will get a lower level initial set H-algebra; if we start with a WinSAlg we will get an initial set H-algebra, see Definition 3.2.9 below).

Now, we state the main result on this section.

Lemma 3.2.7. *Let $i > 0$ and $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ a functor. Then, the following type is inhabited:*

$$\mathsf{LLWinSAlg}_i\ \mathsf{H}$$

i.e. any endofunctor has a lower level weakly initial set H-algebra.

Proof. First, we need to construct $(\mathsf{W}, \mathsf{In}_\mathsf{W}) : \mathsf{SAlg}_i\ \mathsf{H}$.

In the proof of Lemma 2.3.6(i), the least fixpoint was the greatest lower bound of the set of all prefixpoints of the monotone function. So, we will define W to be a weak limit for the family of set H-algebras at level $i-1$ (remember, an H-algebra is a generalization to the notion of prefixpoint). Notice that we can diminish level i by one because $i > 0$.

The family of set H-algebras at level $i-1$ is represented by the first projection function:

$$\mathsf{pr}_1 : (\mathsf{SAlg}_{i-1}\ \mathsf{H}) \to \mathcal{U}_{i-1}$$

where type $\mathsf{SAlg}_{i-1}\ \mathsf{H}$ is acting as the "collection of indices".[7]

Now, $(\mathsf{SAlg}_{i-1}\ \mathsf{H}) : \mathcal{U}_i$, because:

- We know $\mathbf{Set}_{i-1}$ is defined as $\sum_{\mathsf{Y}:\mathcal{U}_{i-1}} \mathsf{IsSet}\ \mathsf{Y}$. So, by Remark 1.5.29 and the universe introduction rule on page 8, type $\mathbf{Set}_{i-1}$ is at level $\max\{i, i-1\} = i$.
- For every $\mathsf{A} : \mathbf{Set}_{i-1}$, type $(\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$ is at level $\max\{i, i-1\} = i$, since $\mathsf{H}\ \mathsf{A} : \mathcal{U}_i$ and $\mathsf{A} : \mathcal{U}_{i-1}$.
- Therefore, type $\mathsf{SAlg}_{i-1}\ \mathsf{H} \equiv \sum_{\mathsf{A}:\mathbf{Set}_{i-1}} (\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$ is at level $\max\{i, i\} = i$.

So, by Lemma 3.2.3, family pr_1 has a weak limit at level $\max\{i, i-1\} = i$:

$$((\mathsf{W}, \mathsf{f}), \mathsf{q}) : \mathsf{WLimit}_i\ \mathsf{pr}_1 \tag{3.3}$$

Hence, W corresponds to type (see proof of Lemma 3.2.3):

$$\prod_{\mathsf{A}:\mathsf{SAlg}_{i-1}\ \mathsf{H}} \mathsf{pr}_1\ \mathsf{A}$$

and (W, f) is a cone over pr_1.

Claim 1: W is a set.

By Lemma 1.8.18, it is enough to prove that $\mathsf{pr}_1\ \mathsf{A}$ is a set for every $\mathsf{A} : \mathsf{SAlg}_{i-1}\ \mathsf{H}$. But $\mathsf{pr}_1\ \mathsf{A}$ is a set by definition of $\mathsf{SAlg}_{i-1}\ \mathsf{H}$.

This proves the claim.

Now, we need to construct a function $\mathsf{In}_\mathsf{W} : (\mathsf{H}\ \mathsf{W}) \to \mathsf{W}$.

Since W is a weak limit for pr_1, if we can construct a proof that $\mathsf{H}\ \mathsf{W}$ is a cone over pr_1, there will be a function of type $(\mathsf{H}\ \mathsf{W}) \to \mathsf{W}$.

Claim 2: $\mathsf{H}\ \mathsf{W}$ forms a cone over pr_1.

We need to build a function of type $\prod_{\mathsf{Y}:\mathsf{SAlg}_{i-1}\ \mathsf{H}} (\mathsf{H}\ \mathsf{W}) \to (\mathsf{pr}_1\ \mathsf{Y})$, which can be rewritten as:

$$\prod_{(\mathsf{A},\mathsf{In}_\mathsf{A}):\mathsf{SAlg}_{i-1}\ \mathsf{H}} (\mathsf{H}\ \mathsf{W}) \to \mathsf{A}$$

by Convention 1.5.27.

[7] Strictly speaking, the first projection function has type $(\mathsf{SAlg}_{i-1}\ \mathsf{H}) \to \mathbf{Set}_{i-1}$, but we are invoking Convention 1.8.10, so that we *can* interpret the projection function as having type $(\mathsf{SAlg}_{i-1}\ \mathsf{H}) \to \mathcal{U}_{i-1}$.

So, let $(\mathsf{A}, \mathsf{In}_\mathsf{A}) : \mathsf{SAlg}_{i-1}\ \mathsf{H}$. Consider the composition:

$$\mathsf{H}\ \mathsf{W} \xrightarrow{\mathsf{map}^\mathsf{H}\ (\mathsf{f}\ (\mathsf{A},\mathsf{In}_\mathsf{A}))} \mathsf{H}\ \mathsf{A} \xrightarrow{\mathsf{In}_\mathsf{A}} \mathsf{A}$$

where function $\mathsf{f}\ (\mathsf{A}, \mathsf{In}_\mathsf{A}) : \mathsf{W} \to \mathsf{A}$ exists, because (W, f) is a cone over pr_1. Observe we can apply the map^H function, since $\mathsf{W} : \mathcal{U}_i$ and $\mathsf{A} : \mathcal{U}_{i-1}$ (and hence, $\mathsf{A} : \mathcal{U}_i$ by the cumulative universe rule).

This proves the claim.

Therefore, since W is a weak limit, by Claim 2 there is a function $\mathsf{In}_\mathsf{W} : (\mathsf{H}\ \mathsf{W}) \to \mathsf{W}$ satisfying:

$$\prod_{(\mathsf{A},\mathsf{In}_\mathsf{A}):\mathsf{SAlg}_{i-1}\ \mathsf{H}} \prod_{\mathsf{y}:\mathsf{H}\ \mathsf{W}} \mathsf{In}_\mathsf{A}\ (\mathsf{map}^\mathsf{H}\ (\mathsf{f}\ (\mathsf{A}, \mathsf{In}_\mathsf{A}))\ \mathsf{y}) = \mathsf{f}\ (\mathsf{A}, \mathsf{In}_\mathsf{A})\ (\mathsf{In}_\mathsf{W}\ \mathsf{y}) \tag{3.4}$$

We have constructed $(\mathsf{W}, \mathsf{In}_\mathsf{W}) : \mathsf{SAlg}_i\ \mathsf{H}$. It remains to show that the following type is inhabited:

$$\prod_{(\mathsf{B},\mathsf{In}_\mathsf{B}):\mathsf{SAlg}_{i-1}\ \mathsf{H}} \mathsf{AlgMor}\ \mathsf{W}\ \mathsf{B} \tag{3.5}$$

Let $(\mathsf{B}, \mathsf{In}_\mathsf{B}) : \mathsf{SAlg}_{i-1}\ \mathsf{H}$. We have to construct an H-algebra morphism from $(\mathsf{W}, \mathsf{In}_\mathsf{W})$ to $(\mathsf{B}, \mathsf{In}_\mathsf{B})$.

But (W, f) is a cone over pr_1 and $(\mathsf{B}, \mathsf{In}_\mathsf{B})$ is in the family, which means we have $\mathsf{f}\ (\mathsf{B}, \mathsf{In}_\mathsf{B}) : \mathsf{W} \to (\mathsf{pr}_1\ (\mathsf{B}, \mathsf{In}_\mathsf{B}))$, or equivalently, $\mathsf{f}\ (\mathsf{B}, \mathsf{In}_\mathsf{B}) : \mathsf{W} \to \mathsf{B}$.

To show that $\mathsf{f}\ (\mathsf{B}, \mathsf{In}_\mathsf{B})$ is a morphism, we need to show that the following diagram commutes:

$$\begin{array}{ccc} \mathsf{H}\ \mathsf{W} & \xrightarrow{\mathsf{map}^\mathsf{H}\ (\mathsf{f}\ (\mathsf{B},\mathsf{In}_\mathsf{B}))} & \mathsf{H}\ \mathsf{B} \\ {\scriptstyle \mathsf{In}_\mathsf{W}}\downarrow & & \downarrow{\scriptstyle \mathsf{In}_\mathsf{B}} \\ \mathsf{W} & \xrightarrow[\mathsf{f}\ (\mathsf{B},\mathsf{In}_\mathsf{B})]{} & \mathsf{B} \end{array}$$

Let $\mathsf{y} : \mathsf{H}\ \mathsf{W}$. But this diagram is type (3.4) instantiated with $(\mathsf{B}, \mathsf{In}_\mathsf{B})$ and y.

□

Before ending this section, we have a remark regarding Lemma 3.2.7.

Remark 3.2.8. The reader may ask why the LLWinSAlg constructed by Lemma 3.2.7 is not already a lower level initial set H-algebra (see Definition 3.2.9 below) instead of just *weakly* initial. The reader may suspect that the problem resides on our use of *weak* limits instead of the stronger notion of limit.

However, strengthening (3.3) from a weak limit into a limit *seems* to be useless, because a limit will construct unique functions *into* type W, while type (3.5) will require to construct a unique function *out* of type W, if we want an initial algebra.

Nevertheless, the author accepts that there may be some non-obvious way in which initiality will follow if we use a limit instead of a weak limit, but he was unable to see this.

In spite of this, there is a way to construct a (lower level) initial set H-algebra out of a (lower level) weakly initial set H-algebra. This is the subject of the following section.

▲

3.2.2 Refining into a (lower level) initial set algebra

The previous section showed that any endofunctor has a LLWinSAlg (the existence of a WinSAlg for an arbitrary endofunctor will be answered in the negative, see Corollary 3.2.26). This section will explore if the result can be strengthened to an initial set H-algebra.

In particular, this section will show that if we are given a LLWinSAlg for some functor H, then we can refine it into a lower level initial set H-algebra. But also, if we are given a WinSAlg for some functor H, then we can refine it into an initial set H-algebra.

First, we need to define the notions of "initial algebra" this section will use.

Definition 3.2.9 ((Lower level) initial set H-algebra)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor.

(i) Let $0 < k \leq i$ be a universe level. Any term of type:

$$\mathsf{LLInSAlg}_k\ \mathsf{H} :\equiv \sum_{(\mathsf{W},\mathsf{In}_\mathsf{W}):\mathsf{SAlg}_k\ \mathsf{H}} \prod_{(\mathsf{B},\mathsf{In}_\mathsf{B}):\mathsf{SAlg}_{k-1}\ \mathsf{H}} \mathsf{IsContr}\ (\mathsf{AlgMor}\ \mathsf{W}\ \mathsf{B})$$

will be called a *lower level initial set* H-*algebra* (*LLInSAlg* for short) at level k.

(ii) Let $k \leq i$ be a universe level. Any term of type:

$$\mathsf{InSAlg}_k\ \mathsf{H} :\equiv \sum_{(\mathsf{W},\mathsf{In}_\mathsf{W}):\mathsf{SAlg}_k\ \mathsf{H}} \prod_{(\mathsf{B},\mathsf{In}_\mathsf{B}):\mathsf{SAlg}_k\ \mathsf{H}} \mathsf{IsContr}\ (\mathsf{AlgMor}\ \mathsf{W}\ \mathsf{B})$$

will be called an *initial set* H-*algebra* (*InSAlg* for short) at level k. Notice this is just Definition 1.9.15, but restricted to sets.

▲

Convention 3.2.10. We will write lower level initial set H-algebras as $(\mathsf{W}, \mathsf{In}_\mathsf{W}, \mathsf{mc})$, instead of $((\mathsf{W}, \mathsf{In}_\mathsf{W}), \mathsf{mc})$. The same applies to initial set H-algebras.

▲

Definition 3.2.9 is like Definition 3.2.5, with the exception that algebra morphisms are asserted to be *unique*, i.e. type $\mathsf{AlgMor}\ \mathsf{W}\ \mathsf{B}$ is contractible (see Definition 1.8.1).

It is fairly obvious that any InSAlg is a LLInSAlg (just by unfolding definitions), but the other way around is not necessarily true. Although both concepts look superficially the same, LLInSAlgs are *weaker* than InSAlgs, as it will shown that LLInSAlgs are sometimes InSAlgs (see Section 3.2.4) but sometimes they are not (see comments after Theorem 3.2.25).

To support the claim that LLInSAlgs are weaker than InSAlgs, Corollary 3.2.22 will show that *any* endofunctor has a LLInSAlg, while Theorem 3.2.25 will show that there are endofunctors without an InSAlg.

Another way to support this "weakness claim" is through the following lemma, due to Lambek (see Lemma 2.5.5 in [17]). This lemma can be proved only for InSAlgs, but not for LLInSAlgs, as we will explain shortly.

Lemma 3.2.11 (Lambek)**.** *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be set-preserving*[8] *functor. Let* $k \leq i$ *be a universe level, and* $(\mathsf{A}, \mathsf{In}_\mathsf{A}, \mathsf{mc}) : \mathsf{InSAlg}_k\ \mathsf{H}$ *an initial set* H-*algebra. Then,* $\mathsf{H}\ \mathsf{A}$ *and* A *are equivalent types. In other words, the following type is inhabited:*

$$\mathsf{H}\ \mathsf{A} \simeq \mathsf{A}$$

Proof. By Lemma 1.6.18, to prove that $\mathsf{H}\ \mathsf{A}$ and A are equivalent, it is enough to construct two mutually inverse functions $(\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$ and $\mathsf{A} \to (\mathsf{H}\ \mathsf{A})$.

We take $\mathsf{In}_\mathsf{A} : (\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$ as the first function. It remains to define the second function.

Observe that $\mathsf{H}\ \mathsf{A}$ is at universe level i. Also, $\mathsf{H}\ \mathsf{A}$ is a set, since A is a set by definition, and H preserves sets.

To make $\mathsf{H}\ \mathsf{A}$ a set H-algebra, we require an "In" function. Consider:

$$\mathsf{map}^\mathsf{H}\ \mathsf{In}_\mathsf{A} : (\mathsf{H}\ (\mathsf{H}\ \mathsf{A})) \to (\mathsf{H}\ \mathsf{A})$$

Therefore, we can form the set H-algebra $(\mathsf{H}\ \mathsf{A}, \mathsf{map}^\mathsf{H}\ \mathsf{In}_\mathsf{A})$ at level i.

This means we can apply term mc on algebra $(\mathsf{H}\ \mathsf{A}, \mathsf{map}^\mathsf{H}\ \mathsf{In}_\mathsf{A})$ to get a morphism $\mathsf{A} \to (\mathsf{H}\ \mathsf{A})$. More specifically, we have a morphism:[9]

$$\mathsf{h}_\mathsf{m} :\equiv \mathsf{center}\ (\mathsf{mc}\ (\mathsf{H}\ \mathsf{A}, \mathsf{map}^\mathsf{H}\ \mathsf{In}_\mathsf{A})) : \mathsf{AlgMor}\ \mathsf{A}\ (\mathsf{H}\ \mathsf{A})$$

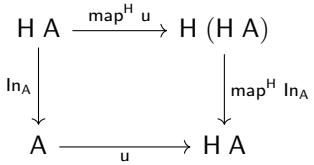
Therefore, we can define our second function as:

$$\mathsf{u} :\equiv \mathsf{pr}_1\ \mathsf{h}_\mathsf{m} : \mathsf{A} \to (\mathsf{H}\ \mathsf{A})$$

[8] See Definition 1.9.9.

[9] Since $\mathsf{mc}\ (\mathsf{H}\ \mathsf{A}, \mathsf{map}^\mathsf{H}\ \mathsf{In}_\mathsf{A}) : \mathsf{IsContr}\ (\mathsf{AlgMor}\ \mathsf{A}\ (\mathsf{H}\ \mathsf{A}))$, we can obtain its center of contraction. See Definition 1.8.1.

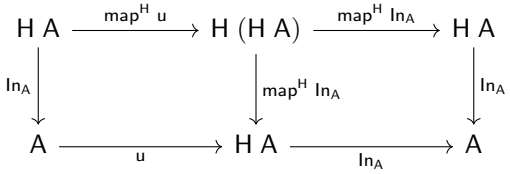
Notice that we also have a proof ($\mathsf{pr}_2\ \mathsf{h_m}$) stating that the following diagram commutes:



(3.6)

Now, it remains to show that $\mathsf{In_A}$ and u are inverses of each other.

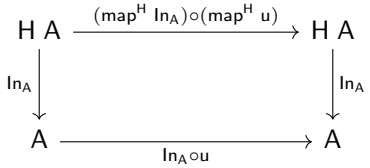
- Type $\prod_{\mathsf{w:A}} \mathsf{In_A}\ (\mathsf{u}\ \mathsf{w}) = \mathsf{w}$ is inhabited.
  We have the two commutative squares:



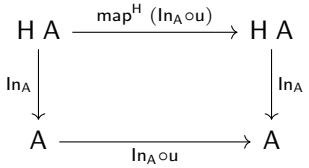
  where the square on the left is Diagram (3.6), and the square on the right is the composition:

$$\mathsf{H\ (H\ A)} \xrightarrow{\mathsf{map^H\ In_A}} \mathsf{H\ A} \xrightarrow{\mathsf{In_A}} \mathsf{A}$$

  written twice on the sides of the square (hence the right square trivially commutes).
  This means that the "external" square commutes:



  which, by functorial mapping of H (see Definition 1.9.6), is equivalent to the commutative diagram:



  This last diagram claims that function $\mathsf{In_A} \circ \mathsf{u}$ is an algebra morphism from A to A. But the identity function $\mathsf{id_A}$ is also an algebra morphism from A to A (Lemma 1.9.14). Since A is an initial set H-algebra, we must have that $\mathsf{In_A} \circ \mathsf{u} = \mathsf{id_A}$ holds, because morphisms from A to A are unique.

- Type $\prod_{\mathsf{w:H\ A}} \mathsf{u}\ (\mathsf{In_A}\ \mathsf{w}) = \mathsf{w}$ is inhabited.
  Let $\mathsf{w} : \mathsf{H\ A}$. We have the following sequence of identifications:

$$\begin{aligned}
\mathsf{u}\ (\mathsf{In_A}\ \mathsf{w}) &\overset{(1)}{=} \mathsf{map^H\ In_A}\ (\mathsf{map^H\ u\ w}) \\
&\overset{(2)}{=} \mathsf{map^H}\ (\mathsf{In_A} \circ \mathsf{u})\ \mathsf{w} \\
&\overset{(3)}{=} \mathsf{map^H\ id_A\ w} \\
&\overset{(4)}{=} \mathsf{id}_{(\mathsf{H\ A})}\mathsf{w} \\
&\overset{(5)}{=} \mathsf{w}
\end{aligned}$$

where (1) follows by commutativity of Diagram (3.6), (2) by functorial mapping of H, (3) by the previous case and function extensionality,[10] (4) by functorial mapping of H on the identity function, and (5) by definition of the identity function.

□

Lambek's Lemma claims that an initial set H-algebra is a "fixpoint" for functor H. This becomes clear if we apply univalence to the conclusion of Lambek's Lemma, because we will have the equation $\mathsf{H}\ \mathsf{A} = \mathsf{A}$ (i.e. A is a fixpoint for H).

If we try to repeat Lambek's Lemma for a LLInSAlg $(\mathsf{A}, \mathsf{In}_\mathsf{A}, \mathsf{mc})$, we will get stuck while defining function $\mathsf{A} \to (\mathsf{H}\ \mathsf{A})$. The reason is that type $\mathsf{H}\ \mathsf{A}$ is at level i. Therefore, we will not be able to apply mc on $\mathsf{H}\ \mathsf{A}$ because mc will expect an algebra at level $i-1$. Later, we will construct a LLInSAlg for which the conclusion of Lambek's Lemma does not hold (see Corollary 3.2.27). Therefore, LLInSAlgs are weaker than InSAlgs because they fail to be "fixpoints" in general.

Before starting the refinement process, we need a couple of definitions and lemmas.

Given a set H-algebra $(\mathsf{A}, \mathsf{In}_\mathsf{A})$ and a predicate $\mathsf{P} : \mathsf{A} \to \mathbf{Prop}_i$, it will be useful to restrict the domain of In_A to only those terms in A satisfying the predicate P. This is the subject of the following definition.

Definition 3.2.12 (Property In function)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k, m \leq i$ universe levels, $(\mathsf{A}, \mathsf{In}_\mathsf{A}) : \mathsf{SAlg}_k\ \mathsf{H}$ a set H-algebra, and $\mathsf{P} : \mathsf{A} \to \mathbf{Prop}_m$ a predicate on A. Then, the function:

$$\mathsf{In}_\mathsf{A}^\mathsf{P} : \left(\mathsf{H} \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right) \to \mathsf{A}$$

defined by composition as:

$$\left(\mathsf{H} \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}\right) \xrightarrow{\mathsf{map}^\mathsf{H}\ \mathsf{pr}_1} \mathsf{H}\ \mathsf{A} \xrightarrow{\mathsf{In}_\mathsf{A}} \mathsf{A}$$

will be called the P-*property In* (or just *property In* when P is clear from context) for $(\mathsf{A}, \mathsf{In}_\mathsf{A})$.

▲

Notice we can apply functor H on type $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$, because $\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}$ is at level $\max\{k, m\} \leq i$.

The "property In" function has the following properties.

Lemma 3.2.13. *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor,* $k, l, m, n \leq i$ *universe levels,* $(\mathsf{A}, \mathsf{In}_\mathsf{A}) : \mathsf{SAlg}_k\ \mathsf{H}$ *and* $(\mathsf{B}, \mathsf{In}_\mathsf{B}) : \mathsf{SAlg}_l\ \mathsf{H}$ *two set* H*-algebras,* $\mathsf{P} : \mathsf{A} \to \mathbf{Prop}_m$ *and* $\mathsf{Q} : \mathsf{B} \to \mathbf{Prop}_n$ *two predicates,* $\mathsf{g} : \mathsf{A} \to \mathsf{B}$ *a function, and a function:*

$$\mathsf{h} : \prod_{\mathsf{w}:\mathsf{A}} (\mathsf{P}\ \mathsf{w}) \to (\mathsf{Q}\ (\mathsf{g}\ \mathsf{w}))$$

If g *is an* H*-algebra morphism, then the following type is inhabited:*[11]

$$\prod_{\mathsf{y}:\mathsf{H} \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w}} \mathsf{g}\ (\mathsf{In}_\mathsf{A}^\mathsf{P}\ \mathsf{y}) = \mathsf{In}_\mathsf{B}^\mathsf{Q}\ (\mathsf{map}^\mathsf{H}\ (\Sigma_{\mathsf{map}}\ \mathsf{g}\ \mathsf{h})\ \mathsf{y})$$

Or equivalently, if the following diagram commutes:

$$\begin{array}{ccc} \mathsf{H}\ \mathsf{A} & \xrightarrow{\mathsf{map}^\mathsf{H}\ \mathsf{g}} & \mathsf{H}\ \mathsf{B} \\ {\scriptstyle \mathsf{In}_\mathsf{A}}\downarrow & & \downarrow{\scriptstyle \mathsf{In}_\mathsf{B}} \\ \mathsf{A} & \xrightarrow[\mathsf{g}]{} & \mathsf{B} \end{array}$$

[10] In more detail, $\prod_{\mathsf{z}:\mathsf{A}} \mathsf{In}_\mathsf{A}\ (\mathsf{u}\ \mathsf{z}) = \mathsf{z}$ holds by the previous case. Then, by function extensionality we have a proof $\mathsf{q} : \mathsf{In}_\mathsf{A} \circ \mathsf{u} = \mathsf{id}_\mathsf{A}$. Therefore, we have $\mathsf{ap}_\mathsf{R}\ \mathsf{q} : \mathsf{map}^\mathsf{H}\ (\mathsf{In}_\mathsf{A} \circ \mathsf{u})\ \mathsf{w} = \mathsf{map}^\mathsf{H}\ \mathsf{id}_\mathsf{A}\ \mathsf{w}$, where R is the function $\lambda \mathsf{f} : \mathsf{A} \to \mathsf{A}.\ \mathsf{map}^\mathsf{H}\ \mathsf{f}\ \mathsf{w}$.

[11] The Σ_{map} function was defined in Lemma 1.5.28.

then the following diagram commutes:

$$\begin{array}{ccc} \left(\mathsf{H}\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}\right) & \xrightarrow{\mathsf{map}^{\mathsf{H}}\,(\Sigma_{\mathsf{map}}\,\mathsf{g}\,\mathsf{h})} & \left(\mathsf{H}\sum_{\mathsf{w}:\mathsf{B}}\mathsf{Q}\,\mathsf{w}\right) \\ \downarrow{\scriptstyle \mathsf{In}^{\mathsf{P}}_{\mathsf{A}}} & & \downarrow{\scriptstyle \mathsf{In}^{\mathsf{Q}}_{\mathsf{B}}} \\ \mathsf{A} & \xrightarrow[\mathsf{g}]{} & \mathsf{B} \end{array}$$

Proof. First, we prove that the following diagram commutes:

$$\begin{array}{ccc} \left(\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}\right) & \xrightarrow{\Sigma_{\mathsf{map}}\,\mathsf{g}\,\mathsf{h}} & \left(\sum_{\mathsf{w}:\mathsf{B}}\mathsf{Q}\,\mathsf{w}\right) \\ \downarrow{\scriptstyle \mathsf{pr}_1} & & \downarrow{\scriptstyle \mathsf{pr}_1} \\ \mathsf{A} & \xrightarrow[\mathsf{g}]{} & \mathsf{B} \end{array} \tag{3.7}$$

Let $(\mathsf{a},\mathsf{b}) : \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\,\mathsf{w}$. Then, by definition of $\Sigma_{\mathsf{map}}\,\mathsf{g}\,\mathsf{h}$ and pr_1, we have:

$$\begin{aligned} \mathsf{pr}_1\,(\Sigma_{\mathsf{map}}\,\mathsf{g}\,\mathsf{h}\,(\mathsf{a},\mathsf{b})) &\equiv \mathsf{pr}_1\,(\mathsf{g}\,\mathsf{a},\mathsf{h}\,\mathsf{a}\,\mathsf{b}) \\ &\equiv \mathsf{g}\,\mathsf{a} \\ &\equiv \mathsf{g}\,(\mathsf{pr}_1\,(\mathsf{a},\mathsf{b})) \end{aligned}$$

But H is a functor, which means that Diagram (3.7) passes to H:[12]

$$\begin{array}{ccc} \left(\mathsf{H}\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}\right) & \xrightarrow{\mathsf{map}^{\mathsf{H}}\,(\Sigma_{\mathsf{map}}\,\mathsf{g}\,\mathsf{h})} & \left(\mathsf{H}\sum_{\mathsf{w}:\mathsf{B}}\mathsf{Q}\,\mathsf{w}\right) \\ \downarrow{\scriptstyle \mathsf{map}^{\mathsf{H}}\,\mathsf{pr}_1} & & \downarrow{\scriptstyle \mathsf{map}^{\mathsf{H}}\,\mathsf{pr}_1} \\ \mathsf{H}\,\mathsf{A} & \xrightarrow[\mathsf{map}^{\mathsf{H}}\,\mathsf{g}]{} & \mathsf{H}\,\mathsf{B} \end{array}$$

i.e. given $\mathsf{w} : \mathsf{H}\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}$, we have:

$$\begin{aligned} \mathsf{map}^{\mathsf{H}}\,\mathsf{g}\,(\mathsf{map}^{\mathsf{H}}\,\mathsf{pr}_1\,\mathsf{w}) &\overset{(1)}{=} \mathsf{map}^{\mathsf{H}}\,(\mathsf{g}\circ\mathsf{pr}_1)\,\mathsf{w} \\ &\overset{(2)}{=} \mathsf{map}^{\mathsf{H}}\,(\mathsf{pr}_1\circ(\Sigma_{\mathsf{map}}\,\mathsf{g}\,\mathsf{h}))\,\mathsf{w} \\ &\overset{(3)}{=} \mathsf{map}^{\mathsf{H}}\,\mathsf{pr}_1\,(\mathsf{map}^{\mathsf{H}}\,(\Sigma_{\mathsf{map}}\,\mathsf{g}\,\mathsf{h})\,\mathsf{w}) \end{aligned}$$

where (1) follows by functorial mapping of H, (2) by the commutativity of Diagram (3.7) and function extensionality, and (3) by functorial mapping of H.

Now, since g is a morphism, we have that both squares commute:

$$\begin{array}{ccc} \left(\mathsf{H}\sum_{\mathsf{w}:\mathsf{A}}\mathsf{P}\,\mathsf{w}\right) & \xrightarrow{\mathsf{map}^{\mathsf{H}}\,(\Sigma_{\mathsf{map}}\,\mathsf{g}\,\mathsf{h})} & \left(\mathsf{H}\sum_{\mathsf{w}:\mathsf{B}}\mathsf{Q}\,\mathsf{w}\right) \\ \downarrow{\scriptstyle \mathsf{map}^{\mathsf{H}}\,\mathsf{pr}_1} & & \downarrow{\scriptstyle \mathsf{map}^{\mathsf{H}}\,\mathsf{pr}_1} \\ \mathsf{H}\,\mathsf{A} & \xrightarrow[\mathsf{map}^{\mathsf{H}}\,\mathsf{g}]{} & \mathsf{H}\,\mathsf{B} \\ \downarrow{\scriptstyle \mathsf{In}_{\mathsf{A}}} & & \downarrow{\scriptstyle \mathsf{In}_{\mathsf{B}}} \\ \mathsf{A} & \xrightarrow[\mathsf{g}]{} & \mathsf{B} \end{array}$$

[12] Observe that all types in Diagram (3.7) will be at a level less or equal to i. So, H can be applied on them.

Therefore, the external square commutes:

$$\begin{array}{ccc} \left(\mathsf{H} \sum_{\mathsf{w:A}} \mathsf{P\ w}\right) & \xrightarrow{\mathsf{map^H}\ (\Sigma_{\mathsf{map}}\ \mathsf{g\ h})} & \left(\mathsf{H} \sum_{\mathsf{w:B}} \mathsf{Q\ w}\right) \\ \downarrow{\scriptstyle \mathsf{In_A} \circ (\mathsf{map^H}\ \mathsf{pr_1})} & & \downarrow{\scriptstyle \mathsf{In_B} \circ (\mathsf{map^H}\ \mathsf{pr_1})} \\ \mathsf{A} & \xrightarrow[\mathsf{g}]{} & \mathsf{B} \end{array} \tag{3.8}$$

But, by definition of the "property In" function, Diagram (3.8) is precisely:

$$\begin{array}{ccc} \left(\mathsf{H} \sum_{\mathsf{w:A}} \mathsf{P\ w}\right) & \xrightarrow{\mathsf{map^H}\ (\Sigma_{\mathsf{map}}\ \mathsf{g\ h})} & \left(\mathsf{H} \sum_{\mathsf{w:B}} \mathsf{Q\ w}\right) \\ \downarrow{\scriptstyle \mathsf{In}_{\mathsf{A}}^{\mathsf{P}}} & & \downarrow{\scriptstyle \mathsf{In}_{\mathsf{B}}^{\mathsf{Q}}} \\ \mathsf{A} & \xrightarrow[\mathsf{g}]{} & \mathsf{B} \end{array}$$

□

Lemma 3.2.13 states that function g still "behaves" like an algebra morphism when the domains of $\mathsf{In_A}$ and $\mathsf{In_B}$ are restricted to the subsets induced by predicates P and Q.

Another lemma about the "property In" function expresses what happens when predicate $\mathsf{P} : \mathsf{A} \to \mathbf{Prop}_i$ picks all elements in algebra A, i.e. when P is the constant function $(\lambda \mathsf{w} : \mathsf{A}.\ \mathbb{1})$ (by Lemma 1.8.17, $\mathbb{1}$ is a proposition).

Lemma 3.2.14. *Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k \leq i$ a universe level, and $(\mathsf{A}, \mathsf{In_A}) : \mathsf{SAlg}_k\ \mathsf{H}$ a set H-algebra. Then, the following type is inhabited.*[13]

$$\prod_{\mathsf{y:H} \sum_{\mathsf{w:A}} \mathbb{1}} \mathsf{id_A}\ (\mathsf{In}_{\mathsf{A}}^{\lambda \mathsf{w}.\ \mathbb{1}}\ \mathsf{y}) = \mathsf{In_A}\ (\mathsf{map^H}\ \mathsf{pr_1}\ \mathsf{y})$$

Or equivalently, the following diagram commutes:

$$\begin{array}{ccc} \left(\mathsf{H} \sum_{\mathsf{w:A}} \mathbb{1}\right) & \xrightarrow{\mathsf{map^H}\ \mathsf{pr_1}} & \mathsf{H\ A} \\ \downarrow{\scriptstyle \mathsf{In}_{\mathsf{A}}^{\lambda \mathsf{w}.\ \mathbb{1}}} & & \downarrow{\scriptstyle \mathsf{In_A}} \\ \mathsf{A} & \xrightarrow[\mathsf{id_A}]{} & \mathsf{A} \end{array}$$

Proof. Let $\mathsf{y} : \mathsf{H} \sum_{\mathsf{w:A}} \mathbb{1}$, then we have:

$$\begin{aligned} \mathsf{id_A}\ (\mathsf{In}_{\mathsf{A}}^{\lambda \mathsf{w}.\ \mathbb{1}}\ \mathsf{y}) &\equiv \mathsf{In}_{\mathsf{A}}^{\lambda \mathsf{w}.\ \mathbb{1}}\ \mathsf{y} \\ &\equiv \mathsf{In_A}\ (\mathsf{map^H}\ \mathsf{pr_1}\ \mathsf{y}) \end{aligned}$$

where the last step follows by definition of $\mathsf{In}_{\mathsf{A}}^{\lambda \mathsf{w}.\ \mathbb{1}}$.

□

Given a LLWinSAlg (or a WinSAlg) A, the refinement process into a LLInSAlg (InSAlg respectively) will consist on the following steps:

1. Filter out those elements in A that cannot be placed in a "canonical form". Denote the resultant type $\mathsf{A_I}$.
2. Prove that $\mathsf{A_I}$ is a set H-algebra.
3. Construct an induction principle for $\mathsf{A_I}$.
4. Use the induction principle to prove the initiality of $\mathsf{A_I}$.

The last step is the only one where we need A to be a LLWinSAlg (or a WinSAlg). All other steps can be performed with the weaker assumption that A is a set H-algebra.

[13] Notice that functor H can be applied on type $\sum_{\mathsf{w:A}} \mathbb{1}$ because $\mathbb{1}$ is at every universe level (see its formation rule on page 40). This means $\sum_{\mathsf{w:A}} \mathbb{1}$ is at level $\max\{k, i\} = i$.

Step 1: Filter out non-canonical elements

Given an arbitrary set H-algebra $(\mathsf{A}, \mathsf{In}_\mathsf{A})$, it may have elements that cannot be placed in the form $\mathsf{In}_\mathsf{A}\ \mathsf{b}$ for some $\mathsf{b} : \mathsf{H}\ \mathsf{A}$. We call these elements *non-canonical*. Non-canonical elements represent elements that cannot be built by a constructor, because function In_A encapsulates the idea of "constructors" for A (see Section 1.9 for an explanation).

The intuition behind an initial algebra is that *all its elements can be built only through the constructors*. Therefore, we need a way to remove non-canonical elements from algebra A.

A first approach would be to define type:

$$\sum_{\mathsf{w}:\mathsf{A}} \sum_{\mathsf{b}:\mathsf{H}\ \mathsf{A}} \mathsf{w} = \mathsf{In}_\mathsf{A}\ \mathsf{b}$$

as the subset of canonical elements.

The problem with this definition is that even if w can be placed in the form $\mathsf{In}_\mathsf{A}\ \mathsf{b}$ for some $\mathsf{b} : \mathsf{H}\ \mathsf{A}$, term b itself may have a non-canonical element as a subterm. Therefore, we need a way to build the subset of canonical elements from the "bottom up", i.e. starting with the empty set, we build the next subset by adding all the base constructors, then we build the next subset by applying the inductive constructors on the previous subset, and so on. This will make sure that all elements in the set are built up from canonical elements. In other words, we require this set to be defined inductively. To accomplish this, we will use the results of Chapter 2.

Since subsets can be identified with predicates, we will use the powertype lattice of Example 2.2.8. We will have to define a monotone function on this complete lattice. Then, the least fixpoint of this function will be the predicate "x is a canonical element".

Let us define the monotone function.

Definition 3.2.15. Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k \leq i$ a universe level, and $(\mathsf{A}, \mathsf{In}_\mathsf{A}) : \mathsf{SAlg}_k\ \mathsf{H}$ a set H-algebra. We define a function with type $(\mathsf{A} \to \mathbf{Prop}_i) \to (\mathsf{A} \to \mathbf{Prop}_i)$ as:

$$\mathsf{CanStep}^\mathsf{A} :\equiv \lambda(\mathsf{T} : \mathsf{A} \to \mathbf{Prop}_i)(\mathsf{w} : \mathsf{A}).\ \left\| \sum_{\mathsf{y}:\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{T}\ \mathsf{z}} \mathsf{w} = \mathsf{In}_\mathsf{A}^\mathsf{T}\ \mathsf{y} \right\|$$

called the *canonical element step function* (or just *step function* for short) for set H-algebra A.

▲

Notice that type $\sum_{\mathsf{z}:\mathsf{A}} \mathsf{T}\ \mathsf{z}$ is at level $\max\{k, i\} = i$, which means that it can be applied to functor H. Also, type $\sum_{\mathsf{y}:\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{T}\ \mathsf{z}} (\mathsf{w} = \mathsf{In}_\mathsf{A}^\mathsf{T}\ \mathsf{y})$ is at level $\max\{i, k\} = i$.

To understand the $\mathsf{CanStep}^\mathsf{A}$ function, it is useful to interpret it in set-theoretic language, as follows:

$$CanStep^A(T) = \{w \in A \mid \exists y \in H(T).\ w = In_A(y)\} = In_A[H(T)]$$

where $T \subseteq A$. For the explanation, we look at functor H as a mapping between sets, and we identify predicates with subsets of A. Type $\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{T}\ \mathsf{z}$ can be identified with set $H(T)$, and the truncated sigma can be viewed as a classical existential. Notice that $H(T) \subseteq H(A)$. Therefore, function In_A can be applied without restricting its domain to $H(T)$, as we had to do in type theory by using the T-Property In function $\mathsf{In}_\mathsf{A}^\mathsf{T}$.

In other words, $CanStep^A(T)$ takes the image of $H(T)$ under the In_A function. If we imagine set $T \subseteq A$ as a construction step in our iterative process, then $CanStep^A(T)$ applies functor H on set T so that more elements are "added" to T, and then, all elements on the incremented set are placed in a canonical form by using "some constructor" (represented by the application of function In_A). This way, all elements in set $CanStep^A(T)$ will consist of canonical elements built from canonical elements in set T.

We now prove that $\mathsf{CanStep}^\mathsf{A}$ is a monotone function.

Lemma 3.2.16. *Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k \leq i$ a universe level, and $(\mathsf{A}, \mathsf{In}_\mathsf{A}) : \mathsf{SAlg}_k\ \mathsf{H}$ a set H-algebra. Let:*

$$\mathsf{L} :\equiv (\mathsf{A} \to \mathbf{Prop}_i, \subseteq, \mathsf{rp}, \mathsf{ap}, \mathsf{tp}; \mathsf{sup}, \mathsf{sp}, \mathsf{inf}, \mathsf{ip})$$

be the "Powertype" lattice of Example 2.2.8. Then, $\mathsf{CanStep}^\mathsf{A}$ is a monotone function in L.

Proof. Given $\mathsf{P}, \mathsf{Q} : \mathsf{A} \to \mathbf{Prop}_i$ and $\mathsf{p}_1 : \mathsf{P} \subseteq \mathsf{Q}$, we need to prove $\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P} \subseteq \mathsf{CanStep}^{\mathsf{A}}\ \mathsf{Q}$, or equivalently, by definition of $\subseteq$:

$$\prod_{\mathsf{w}:\mathsf{A}} (\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P}\ \mathsf{w}) \to (\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{Q}\ \mathsf{w})$$

but, by definition of $\mathsf{CanStep}^{\mathsf{A}}$, it is enough to prove:

$$\prod_{\mathsf{w}:\mathsf{A}} \left\| \sum_{\mathsf{y}:\mathsf{H}\sum_{\mathsf{z}:\mathsf{A}}\mathsf{P}\,\mathsf{z}} \mathsf{w} = \mathsf{In}^{\mathsf{P}}_{\mathsf{A}}\ \mathsf{y} \right\| \to \left\| \sum_{\mathsf{y}:\mathsf{H}\sum_{\mathsf{z}:\mathsf{A}}\mathsf{Q}\,\mathsf{z}} \mathsf{w} = \mathsf{In}^{\mathsf{Q}}_{\mathsf{A}}\ \mathsf{y} \right\|$$

Let $\mathsf{w} : \mathsf{A}$. By (-1)-truncation recursion (Corollary 1.11.4), it is enough to prove:

$$\left(\sum_{\mathsf{y}:\mathsf{H}\sum_{\mathsf{z}:\mathsf{A}}\mathsf{P}\,\mathsf{z}} \mathsf{w} = \mathsf{In}^{\mathsf{P}}_{\mathsf{A}}\ \mathsf{y} \right) \to \left\| \sum_{\mathsf{y}:\mathsf{H}\sum_{\mathsf{z}:\mathsf{A}}\mathsf{Q}\,\mathsf{z}} \mathsf{w} = \mathsf{In}^{\mathsf{Q}}_{\mathsf{A}}\ \mathsf{y} \right\|$$

So, suppose we have $\mathsf{y} : \mathsf{H} \sum_{\mathsf{z}:\mathsf{A}} \mathsf{P}\ \mathsf{z}$ with $\mathsf{w} = \mathsf{In}^{\mathsf{P}}_{\mathsf{A}}\ \mathsf{y}$. By the (-1)-truncation constructor:

$$|\,|: \left(\sum_{\mathsf{y}:\mathsf{H}\sum_{\mathsf{z}:\mathsf{A}}\mathsf{Q}\,\mathsf{z}} \mathsf{w} = \mathsf{In}^{\mathsf{Q}}_{\mathsf{A}}\ \mathsf{y} \right) \to \left\| \sum_{\mathsf{y}:\mathsf{H}\sum_{\mathsf{z}:\mathsf{A}}\mathsf{Q}\,\mathsf{z}} \mathsf{w} = \mathsf{In}^{\mathsf{Q}}_{\mathsf{A}}\ \mathsf{y} \right\|$$

it is enough to build a term $\mathsf{t} : \mathsf{H} \sum_{\mathsf{z}:\mathsf{A}} \mathsf{Q}\ \mathsf{z}$ satisfying $\mathsf{w} = \mathsf{In}^{\mathsf{Q}}_{\mathsf{A}}\ \mathsf{t}$.

But we have the Σ_{map} function (Lemma 1.5.28):

$$\left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w} \right) \xrightarrow{\Sigma_{\mathsf{map}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{p}_1} \left(\sum_{\mathsf{w}:\mathsf{A}} \mathsf{Q}\ \mathsf{w} \right)$$

where id_{A} is the identity on A, and $\mathsf{p}_1 : \mathsf{P} \subseteq \mathsf{Q}$ is an hypothesis.[14]

Therefore, by applying H on function $(\Sigma_{\mathsf{map}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{p}_1)$, we get:

$$\left(\mathsf{H} \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w} \right) \xrightarrow{\mathsf{map}^{\mathsf{H}}\ (\Sigma_{\mathsf{map}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{p}_1)} \left(\mathsf{H} \sum_{\mathsf{w}:\mathsf{A}} \mathsf{Q}\ \mathsf{w} \right)$$

We know term y is in the domain of this function. Define:

$$\mathsf{t} :\equiv \mathsf{map}^{\mathsf{H}}\ (\Sigma_{\mathsf{map}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{p}_1)\ \mathsf{y}$$

Since id_{A} is a morphism (Lemma 1.9.14), the following diagram commutes by Lemma 3.2.13:

$$\begin{array}{ccc} \left(\mathsf{H} \sum_{\mathsf{w}:\mathsf{A}} \mathsf{P}\ \mathsf{w} \right) & \xrightarrow{\mathsf{map}^{\mathsf{H}}\ (\Sigma_{\mathsf{map}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{p}_1)} & \left(\mathsf{H} \sum_{\mathsf{w}:\mathsf{A}} \mathsf{Q}\ \mathsf{w} \right) \\ \downarrow{\scriptstyle \mathsf{In}^{\mathsf{P}}_{\mathsf{A}}} & & \downarrow{\scriptstyle \mathsf{In}^{\mathsf{Q}}_{\mathsf{A}}} \\ \mathsf{A} & \xrightarrow[\mathsf{id}_{\mathsf{A}}]{} & \mathsf{A} \end{array}$$

Therefore, we have:

$$\begin{aligned} \mathsf{w} &\overset{(1)}{=} \mathsf{In}^{\mathsf{P}}_{\mathsf{A}}\ \mathsf{y} \\ &\overset{(2)}{=} \mathsf{id}_{\mathsf{A}}\ (\mathsf{In}^{\mathsf{P}}_{\mathsf{A}}\ \mathsf{y}) \\ &\overset{(3)}{=} \mathsf{In}^{\mathsf{Q}}_{\mathsf{A}}\ (\mathsf{map}^{\mathsf{H}}\ (\Sigma_{\mathsf{map}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{p}_1)\ \mathsf{y}) \\ &\overset{(4)}{=} \mathsf{In}^{\mathsf{Q}}_{\mathsf{A}}\ \mathsf{t} \end{aligned}$$

[14] Notice that $\mathsf{P} \subseteq \mathsf{Q} \equiv \prod_{\mathsf{z}:\mathsf{A}} (\mathsf{P}\ \mathsf{z}) \to (\mathsf{Q}\ \mathsf{z}) \equiv \prod_{\mathsf{z}:\mathsf{A}} (\mathsf{P}\ \mathsf{z}) \to (\mathsf{Q}\ (\mathsf{id}_{\mathsf{A}}\ \mathsf{z}))$. The last type has the form required by the Σ_{map} function.

where (1) follows by hypothesis on y, (2) by definition of $\mathsf{id_A}$, (3) by commutativity of the diagram, and (4) by definition of t.

□

Therefore, by Knaster-Tarski (Lemma 2.3.6(i)), function $\mathsf{CanStep^A}$ has a least fixpoint in the "Powertype" lattice. This fixpoint corresponds to the predicate "x is a canonical element". With this, we define the set of canonical elements in algebra A as follows.

Definition 3.2.17. Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k \leq i$ a universe level, and $(\mathsf{A}, \mathsf{In_A}) : \mathsf{SAlg}_k\ \mathsf{H}$ a set H-algebra. Then, type:[15]

$$\mathsf{A_I} :\equiv \sum_{\mathsf{w:A}} \mathsf{Lfp\ CanStep^A\ w}$$

will be called the *set of canonical elements* in A.

▲

Because A is a set and $\mathsf{Lfp\ CanStep^A} : \mathsf{A} \to \mathbf{Prop}_i$ (hence, a family of sets by Lemma 1.8.11), we must have that $\mathsf{A_I}$ is a set by closure properties for Σ-types (Lemma 1.8.18).

When A is a WinSAlg (or LLWinSAlg), type $\mathsf{A_I}$ will be our candidate for the InSAlg (LLInSAlg, respectively).

Step 2: Prove $\mathsf{A_I}$ is a set H-algebra

To prove that $\mathsf{A_I}$ is a set H-algebra, we need to construct a function $\mathsf{In_{A_I}} : (\mathsf{H\ A_I}) \to \mathsf{A_I}$.

Lemma 3.2.18. *Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k \leq i$ a universe level, and $(\mathsf{A}, \mathsf{In_A}) : \mathsf{SAlg}_k\ \mathsf{H}$ a set H-algebra. Then, there is a function $\mathsf{In_{A_I}} : (\mathsf{H\ A_I}) \to \mathsf{A_I}$, so that we have the pair:*

$$(\mathsf{A_I}, \mathsf{In_{A_I}}) : \mathsf{SAlg}_i\ \mathsf{H}$$

i.e. $\mathsf{A_I}$ is a set H-algebra.

Proof. Observe that $\mathsf{A_I}$ is at universe level $\max\{k, i\} = i$, because A is at level $k \leq i$ and $\mathsf{Lfp\ CanStep^A\ w}$ is at level i for every $\mathsf{w} : \mathsf{A}$. Also, $\mathsf{A_I}$ is a set by the discussion after Definition 3.2.17. Therefore, it only remains to construct function $\mathsf{In_{A_I}} : (\mathsf{H\ A_I}) \to \mathsf{A_I}$.

Let $\mathsf{y} : \mathsf{H\ A_I}$. We weed to construct a pair $(\mathsf{a}, \mathsf{b}) : \mathsf{A_I}$, where $\mathsf{a} : \mathsf{A}$ and $\mathsf{b} : \mathsf{Lfp\ CanStep^A\ a}$ is a proof that "a is a canonical element".

We have the Property In function:

$$\mathsf{In_A^{Lfp\ CanStep^A}} : (\mathsf{H} \sum_{\mathsf{w:A}} \mathsf{Lfp\ CanStep^A\ w}) \to \mathsf{A}$$

or equivalently, $\mathsf{In_A^{Lfp\ CanStep^A}} : (\mathsf{H\ A_I}) \to \mathsf{A}$, by definition of $\mathsf{A_I}$.

Hence, we can define:

$$\mathsf{a} :\equiv \mathsf{In_A^{Lfp\ CanStep^A}\ y}$$

Now, we need to construct a term $\mathsf{b} : \mathsf{Lfp\ CanStep^A\ a}$ (it is enough to show that $\mathsf{Lfp\ CanStep^A\ a}$ is inhabited).

In the proof for the first part of Knaster-Tarski (Lemma 2.3.6(i)), the least fixpoint was defined as the infimum of all prefixpoints of the monotone function. When Knaster-Tarski is instantiated in the "Powertype" lattice of Example 2.2.8, the infimum function is:

$$\lambda(\mathsf{F} : (\mathsf{A} \to \mathbf{Prop}_i) \to \mathbf{Prop}_l)(\mathsf{w} : \mathsf{A}).\ \mathsf{GPropR}^{-1} \left(\prod_{\mathsf{P:A} \to \mathbf{Prop}_i} (\mathsf{F\ P}) \to (\mathsf{P\ w}) \right) \tag{3.9}$$

where $\mathsf{GPropR} : \mathbf{Prop}_i \to \mathbf{Prop}_{\max\{k, i+1, l\}}$ is a propositional resizing equivalence, and l is an arbitrary universe level independent of levels k and i.

[15] $\mathsf{Lfp\ CanStep^A}$ is notation for the least fixpoint of $\mathsf{CanStep^A}$, see Lemma 2.3.6(i).

Therefore, $\mathsf{Lfp}\ \mathsf{CanStep}^{\mathsf{A}}$ corresponds to the predicate:[16]

$$\lambda \mathsf{w}:\mathsf{A}.\ \mathsf{GPropR}^{-1}\left(\prod_{\mathsf{P}:\mathsf{A}\to\mathbf{Prop}_i} (\mathsf{IsPrefixpoint}\ \mathsf{P}\ \mathsf{CanStep}^{\mathsf{A}}) \to (\mathsf{P}\ \mathsf{w})\right)$$

or equivalently:

$$\lambda \mathsf{w}:\mathsf{A}.\ \mathsf{GPropR}^{-1}\left(\prod_{\mathsf{P}:\mathsf{A}\to\mathbf{Prop}_i} (\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P} \subseteq \mathsf{P}) \to (\mathsf{P}\ \mathsf{w})\right)$$

Hence, type $(\mathsf{Lfp}\ \mathsf{CanStep}^{\mathsf{A}}\ \mathsf{a})$ corresponds to:

$$\mathsf{GPropR}^{-1}\left(\prod_{\mathsf{P}:\mathsf{A}\to\mathbf{Prop}_i} (\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P} \subseteq \mathsf{P}) \to (\mathsf{P}\ \mathsf{a})\right)$$

To prove that this type is inhabited, it is enough to prove (by Lemma 1.8.23):

$$\prod_{\mathsf{P}:\mathsf{A}\to\mathbf{Prop}_i} (\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P} \subseteq \mathsf{P}) \to (\mathsf{P}\ \mathsf{a})$$

In other words, to prove that term a is in the least fixpoint of $\mathsf{CanStep}^{\mathsf{A}}$, we need to prove that term a is in every prefixpoint of $\mathsf{CanStep}^{\mathsf{A}}$.

Let $\mathsf{P}:\mathsf{A}\to\mathbf{Prop}_i$ with $\mathsf{p}_1:\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P} \subseteq \mathsf{P}$.

By definition of $\subseteq$, we have:

$$(\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P} \subseteq \mathsf{P}) \equiv \left(\prod_{\mathsf{z}:\mathsf{A}} (\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P}\ \mathsf{z}) \to (\mathsf{P}\ \mathsf{z})\right)$$

Hence, if we can construct a proof $\mathsf{d}:\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P}\ \mathsf{a}$, then $\mathsf{p}_1\ \mathsf{a}\ \mathsf{d}:\mathsf{P}\ \mathsf{a}$ will be the required proof.

By definition, type $(\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P}\ \mathsf{a})$ is:

$$\left\| \sum_{\mathsf{w}:\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{P}\ \mathsf{z}} \mathsf{a} = \mathsf{In}^{\mathsf{P}}_{\mathsf{A}}\ \mathsf{w} \right\|$$

By the (-1)-truncation constructor:

$$|\ |: \left(\sum_{\mathsf{w}:\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{P}\ \mathsf{z}} \mathsf{a} = \mathsf{In}^{\mathsf{P}}_{\mathsf{A}}\ \mathsf{w}\right) \to \left\| \sum_{\mathsf{w}:\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{P}\ \mathsf{z}} \mathsf{a} = \mathsf{In}^{\mathsf{P}}_{\mathsf{A}}\ \mathsf{w} \right\|$$

it is enough to build a term $\mathsf{t}:\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{P}\ \mathsf{z}$ satisfying $\mathsf{a} = \mathsf{In}^{\mathsf{P}}_{\mathsf{A}}\ \mathsf{t}$.

By $\mathsf{p}_1:\mathsf{CanStep}^{\mathsf{A}}\ \mathsf{P} \subseteq \mathsf{P}$ and the conventional induction principle (Corollary 2.4.1), there is a term:

$$\mathsf{p}_2:\mathsf{Lfp}\ \mathsf{CanStep}^{\mathsf{A}} \subseteq \mathsf{P}$$

which, by definition of $\subseteq$, is equivalent to:

$$\mathsf{p}_2:\prod_{\mathsf{z}:\mathsf{A}} (\mathsf{Lfp}\ \mathsf{CanStep}^{\mathsf{A}}\ \mathsf{z}) \to (\mathsf{P}\ \mathsf{z})$$

[16] The set of all prefixpoints of $\mathsf{CanStep}^{\mathsf{A}}$ is represented by the predicate:

$$(\lambda \mathsf{P}:(\mathsf{A}\to\mathbf{Prop}_i).\ \mathsf{IsPrefixpoint}\ \mathsf{P}\ \mathsf{CanStep}^{\mathsf{A}}):(\mathsf{A}\to\mathbf{Prop}_i)\to\mathbf{Prop}_i$$

So, to compute $\mathsf{Lfp}\ \mathsf{CanStep}^{\mathsf{A}}$, we give this predicate as input to the infimum function (3.9).

By the Σ_{map} function, we have:

$$\left(\sum_{w:A} \mathsf{Lfp}\ \mathsf{CanStep}^A\ w\right) \xrightarrow{\Sigma_{\mathsf{map}}\ \mathsf{id}_A\ p_2} \left(\sum_{w:A} P\ w\right)$$

or equivalently, by definition of A_I:

$$A_I \xrightarrow{\Sigma_{\mathsf{map}}\ \mathsf{id}_A\ p_2} \left(\sum_{w:A} P\ w\right)$$

Therefore, by functorial mapping on H, we get the function:

$$H\ A_I \xrightarrow{\mathsf{map}^H\ (\Sigma_{\mathsf{map}}\ \mathsf{id}_A\ p_2)} \left(H \sum_{w:A} P\ w\right)$$

We know $y : H\ A_I$ by hypothesis. So, define:

$$t :\equiv \mathsf{map}^H\ (\Sigma_{\mathsf{map}}\ \mathsf{id}_A\ p_2)\ y$$

Since id_A is a morphism (Lemma 1.9.14), the following diagram commutes by Lemma 3.2.13:

$$\begin{array}{ccc} H\ A_I & \xrightarrow{\mathsf{map}^H\ (\Sigma_{\mathsf{map}}\ \mathsf{id}_A\ p_2)} & \left(H \sum_{w:A} P\ w\right) \\ {\scriptstyle \mathsf{In}_A^{\mathsf{Lfp}\ \mathsf{CanStep}^A}}\downarrow & & \downarrow{\scriptstyle \mathsf{In}_A^P} \\ A & \xrightarrow[\mathsf{id}_A]{} & A \end{array}$$

Therefore, we have:

$$\begin{aligned} a &\overset{(1)}{=} \mathsf{In}_A^{\mathsf{Lfp}\ \mathsf{CanStep}^A}\ y \\ &\overset{(2)}{=} \mathsf{id}_A\ (\mathsf{In}_A^{\mathsf{Lfp}\ \mathsf{CanStep}^A}\ y) \\ &\overset{(3)}{=} \mathsf{In}_A^P\ (\mathsf{map}^H\ (\Sigma_{\mathsf{map}}\ \mathsf{id}_A\ p_2)\ y) \\ &\overset{(4)}{=} \mathsf{In}_A^P\ t \end{aligned}$$

where (1) follows by definition of term a, (2) by definition of id_A, (3) by commutativity of the diagram, and (4) by definition of t.

□

Step 3: Construct an induction principle for A_I

First, we need a small lemma stating that the first projection function $\mathsf{pr}_1 : A_I \to A$ is an algebra morphism.

Lemma 3.2.19. *Let $H : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k \leq i$ a universe level, and $(A, \mathsf{In}_A) : \mathsf{SAlg}_k\ H$ a set H-algebra. Then, the first projection:*

$$\mathsf{pr}_1 : A_I \to A$$

is an H-algebra morphism. Or equivalently, the following diagram commutes:

$$\begin{array}{ccc} H\ A_I & \xrightarrow{\mathsf{map}^H\ \mathsf{pr}_1} & H\ A \\ {\scriptstyle \mathsf{In}_{A_I}}\downarrow & & \downarrow{\scriptstyle \mathsf{In}_A} \\ A_I & \xrightarrow[\mathsf{pr}_1]{} & A \end{array}$$

Proof. This follows directly by construction of $\mathsf{In}_{\mathsf{A_I}}$ in Lemma 3.2.18.

In more detail, let $\mathsf{w} : \mathsf{H}\ \mathsf{A_I}$. Then, we have by definitions and computation:

$$\begin{aligned}\mathsf{pr_1}\ (\mathsf{In}_{\mathsf{A_I}}\ \mathsf{w}) &\equiv \mathsf{pr_1}\ (\mathsf{In}_{\mathsf{A}}^{\mathsf{Lfp\ CanStep^A}}\ \mathsf{w}, \mathsf{t}) \\ &\equiv \mathsf{In}_{\mathsf{A}}^{\mathsf{Lfp\ CanStep^A}}\ \mathsf{w} \\ &\equiv \mathsf{In}_{\mathsf{A}}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{pr_1}\ \mathsf{w})\end{aligned}$$

where t is some term with type $\mathsf{Lfp\ CanStep^A}\ (\mathsf{In}_{\mathsf{A}}^{\mathsf{Lfp\ CanStep^A}}\ \mathsf{w})$, but we do not care for the actual term.

□

We can now state an induction principle for $\mathsf{A_I}$.

Lemma 3.2.20 (Induction Principle)**.** *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor,* $k, m \leq i$ *universe levels, and* $(\mathsf{A}, \mathsf{In_A}) : \mathsf{SAlg}_k\ \mathsf{H}$ *a set* H*-algebra.*

Let $\mathsf{P} : \mathsf{A_I} \to \mathbf{Prop}_m$ *be a predicate on* $\mathsf{A_I}$*. If the following type is inhabited:*

$$\prod_{\mathsf{T}:\mathsf{A_I}\to\mathbf{Prop}_m} \left(\prod_{\mathsf{w}:\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T\ y}} \mathsf{P}\ (\mathsf{pr_1}\ \mathsf{w}) \right) \to \left(\prod_{\mathsf{w}:\mathsf{H}\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T\ y}} \mathsf{P}\ (\mathsf{In}_{\mathsf{A_I}}^{\mathsf{T}}\ \mathsf{w}) \right) \tag{3.10}$$

then, the following type is inhabited:

$$\prod_{\mathsf{n}:\mathsf{A_I}} \mathsf{P\ n} \tag{3.11}$$

Proof. Let:

$$\mathsf{Hind} : \prod_{\mathsf{T}:\mathsf{A_I}\to\mathbf{Prop}_m} \left(\prod_{\mathsf{w}:\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T\ y}} \mathsf{P}\ (\mathsf{pr_1}\ \mathsf{w}) \right) \to \left(\prod_{\mathsf{w}:\mathsf{H}\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T\ y}} \mathsf{P}\ (\mathsf{In}_{\mathsf{A_I}}^{\mathsf{T}}\ \mathsf{w}) \right)$$

Define:

$$\mathsf{B} :\equiv \left(\lambda \mathsf{w} : \mathsf{A}. \sum_{\mathsf{q}:\mathsf{Lfp\ CanStep^A\ w}} \mathsf{P}\ (\mathsf{w}, \mathsf{q}) \right) : \mathsf{A} \to \mathbf{Prop}_i$$

First, we explain that B is well-typed. Type $\mathsf{Lfp\ CanStep^A\ w}$ is a mere proposition for every $\mathsf{w} : \mathsf{A}$, and $\mathsf{P\ b}$ is a mere proposition for every $\mathsf{b} : \mathsf{A_I}$. Therefore, by closure properties for sigma types (Lemma 1.8.17), $\mathsf{B\ b}$ is a mere proposition for every $\mathsf{b} : \mathsf{A}$. Also, since $\mathsf{Lfp\ CanStep^A\ w}$ is at universe level i for every $\mathsf{w} : \mathsf{A}$, and $\mathsf{P\ b}$ is at universe level m for every $\mathsf{b} : \mathsf{A_I}$, we will have that $\mathsf{B\ b}$ is at universe level $\max\{i, m\} = i$ for every $\mathsf{b} : \mathsf{A}$.

Observe that B is basically P, but seen as a predicate on A instead of $\mathsf{A_I}$.

Now, we prove a claim.

Claim 1: $\mathsf{Lfp\ CanStep^A} \subseteq \mathsf{B}$ is inhabited.

If we can prove that $\mathsf{CanStep^A\ B} \subseteq \mathsf{B}$ is inhabited, the claim will follow by conventional induction (Corollary 2.4.1).

Let $\mathsf{w} : \mathsf{A}$. We need to prove that the following type is inhabited:

$$(\mathsf{CanStep^A\ B\ w}) \to (\mathsf{B\ w})$$

or equivalently, by definition of $\mathsf{CanStep^A}$:

$$\left\| \sum_{\mathsf{y}:\mathsf{H}\sum_{\mathsf{z}:\mathsf{A}} \mathsf{B\ z}} \mathsf{w} = \mathsf{In}_{\mathsf{A}}^{\mathsf{B}}\ \mathsf{y} \right\| \to (\mathsf{B\ w})$$

Since $\mathsf{B}\ \mathsf{w}$ is a mere proposition, by (-1)-truncation recursion it is enough to prove:

$$\left(\sum_{\mathsf{y}:\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{B}\ \mathsf{z}} \mathsf{w} = \mathsf{In}^{\mathsf{B}}_{\mathsf{A}}\ \mathsf{y}\right) \to \mathsf{B}\ \mathsf{w}$$

Let $\mathsf{y}: \mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{B}\ \mathsf{z}$ with $\mathsf{w} = \mathsf{In}^{\mathsf{B}}_{\mathsf{A}}\ \mathsf{y}$. We need to prove that $\mathsf{B}\ \mathsf{w}$ is inhabited.

By Lemma 1.6.23, we know sigma types are "associative". In other words we have an equivalence:

$$\left(\sum_{\mathsf{z}:\sum_{\mathsf{q}:\mathsf{A}} \mathsf{Lfp}\ \mathsf{CanStep}^{\mathsf{A}}\ \mathsf{q}} \mathsf{P}\ \mathsf{z}\right) \xrightarrow{\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{j}} \left(\sum_{\mathsf{z}:\mathsf{A}}\ \sum_{\mathsf{q}:\mathsf{Lfp}\ \mathsf{CanStep}^{\mathsf{A}}\ \mathsf{z}} \mathsf{P}\ (\mathsf{z}, \mathsf{q})\right)$$

or equivalently, by putting back definitions:

$$\left(\sum_{\mathsf{z}:\mathsf{A}_{\mathsf{I}}} \mathsf{P}\ \mathsf{z}\right) \xrightarrow{\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{j}} \left(\sum_{\mathsf{z}:\mathsf{A}} \mathsf{B}\ \mathsf{z}\right)$$

where:

$$\mathsf{j}: \prod_{\mathsf{z}:\mathsf{A}_{\mathsf{I}}} (\mathsf{P}\ \mathsf{z}) \to (\mathsf{B}\ (\mathsf{pr}_1\ \mathsf{z}))$$

is some function (we only care that it exists by Lemma 1.6.23).

Therefore, by Lemma 1.9.10, we have an equivalence:

$$\left(\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}_{\mathsf{I}}} \mathsf{P}\ \mathsf{z}\right) \xrightarrow{\mathsf{map}^{\mathsf{H}}\ (\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{j})} \left(\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{B}\ \mathsf{z}\right)$$

Define as a shorthand:

$$\mathsf{s} :\equiv \mathsf{map}^{\mathsf{H}}\ (\Sigma_{\mathsf{map}}\ \mathsf{pr}_1\ \mathsf{j})$$

Since pr_1 is an H-algebra morphism (Lemma 3.2.19), the following diagram commutes by Lemma 3.2.13:

$$\begin{array}{ccc} \left(\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}_{\mathsf{I}}} \mathsf{P}\ \mathsf{z}\right) & \xrightarrow{\mathsf{s}} & \left(\mathsf{H}\ \sum_{\mathsf{z}:\mathsf{A}} \mathsf{B}\ \mathsf{z}\right) \\ \downarrow{\scriptstyle \mathsf{In}^{\mathsf{P}}_{\mathsf{A}_{\mathsf{I}}}} & & \downarrow{\scriptstyle \mathsf{In}^{\mathsf{B}}_{\mathsf{A}}} \\ \mathsf{A}_{\mathsf{I}} & \xrightarrow[\mathsf{pr}_1]{} & \mathsf{A} \end{array}$$

By hypothesis, we know term y is in the codomain of function s, and since function s is an equivalence, we have that term $\mathsf{s}^{-1}\ \mathsf{y}$ is in the domain of s. Therefore, the following holds:

$$\begin{aligned} \mathsf{pr}_1\ (\mathsf{In}^{\mathsf{P}}_{\mathsf{A}_{\mathsf{I}}}\ (\mathsf{s}^{-1}\ \mathsf{y})) &\overset{(1)}{=} \mathsf{In}^{\mathsf{B}}_{\mathsf{A}}\ (\mathsf{s}\ (\mathsf{s}^{-1}\ \mathsf{y})) \\ &\overset{(2)}{=} \mathsf{In}^{\mathsf{B}}_{\mathsf{A}}\ \mathsf{y} \\ &\overset{(3)}{=} \mathsf{w} \end{aligned} \tag{3.12}$$

where (1) follows by commutativity of the previous diagram, (2) by $\mathsf{s}\ (\mathsf{s}^{-1}\ \mathsf{y}) = \mathsf{y}$ since s is an equivalence, and (3) by hypothesis on y.

Now, by applying Hind with P, the second projection function:

$$\mathsf{pr}_2: \prod_{\mathsf{z}:\sum_{\mathsf{r}:\mathsf{A}_{\mathsf{I}}} \mathsf{P}\ \mathsf{r}} \mathsf{P}\ (\mathsf{pr}_1\ \mathsf{z})$$

and $s^{-1}\ y$, we get:

$$\mathsf{Hind}\ P\ \mathsf{pr}_2\ (s^{-1}\ y) : P\ (\mathsf{In}^P_{A_I}\ (s^{-1}\ y))$$

Define as a shorthand:

$$r :\equiv \mathsf{In}^P_{A_I}\ (s^{-1}\ y) : A_I$$

so that:

$$\mathsf{Hind}\ P\ \mathsf{pr}_2\ (s^{-1}\ y) : P\ r$$

Since $r : A_I$ is a pair, by Lemma 1.6.1, we know:

$$r = (\mathsf{pr}_1\ r,\ \mathsf{pr}_2\ r)$$

Therefore, there is a term:[17]

$$h_1 : P\ (\mathsf{pr}_1\ r,\ \mathsf{pr}_2\ r)$$

But:

$$\mathsf{pr}_2\ r : \mathsf{Lfp}\ \mathsf{CanStep}^A\ (\mathsf{pr}_1\ r)$$

since $r : A_I$, or equivalently, $r : \sum_{z:A} \mathsf{Lfp}\ \mathsf{CanStep}^A\ z$.

This means that:

$$(\mathsf{pr}_2\ r, h_1) : \sum_{q:\mathsf{Lfp}\ \mathsf{CanStep}^A\ (\mathsf{pr}_1\ r)} P\ (\mathsf{pr}_1\ r, q)$$

or equivalently, by definition of B:

$$(\mathsf{pr}_2\ r, h_1) : B\ (\mathsf{pr}_1\ r)$$

But, by (3.12) and definition of r, we have:

$$\begin{aligned} \mathsf{pr}_1\ r &= \mathsf{pr}_1\ (\mathsf{In}^P_{A_I}\ (s^{-1}\ y)) \\ &= w \end{aligned}$$

This means that $B\ w$ is inhabited.

This proves the claim.

To finish the proof, we need to prove that $\prod_{n:A_I} P\ n$ is inhabited.

Let $n : A_I$. This means we have $\mathsf{pr}_1\ n : A$ and $\mathsf{pr}_2\ n : \mathsf{Lfp}\ \mathsf{CanStep}^A\ (\mathsf{pr}_1\ n)$.

By Claim 1, there is a term $k : \mathsf{Lfp}\ \mathsf{CanStep}^A \subseteq B$. By definition of $\subseteq$, this means:

$$k : \prod_{z:A} (\mathsf{Lfp}\ \mathsf{CanStep}^A\ z) \to (B\ z)$$

Therefore:

$$k\ (\mathsf{pr}_1\ n)\ (\mathsf{pr}_2\ n) : B\ (\mathsf{pr}_1\ n)$$

By definition of B, this means we have $q : \mathsf{Lfp}\ \mathsf{CanStep}^A\ (\mathsf{pr}_1\ n)$ such that $P\ (\mathsf{pr}_1\ n, q)$.

But $\mathsf{Lfp}\ \mathsf{CanStep}^A\ (\mathsf{pr}_1\ n)$ is a mere proposition, which means $q = \mathsf{pr}_2\ n$ is inhabited. So, the following type is inhabited:[18]

$$P\ (\mathsf{pr}_1\ n, \mathsf{pr}_2\ n)$$

But we know $n = (\mathsf{pr}_1\ n, \mathsf{pr}_2\ n)$ by Lemma 1.6.1, which means that the following type is inhabited:

$$P\ n$$

□

[17] In more detail, if we have the proof $q : r = (\mathsf{pr}_1\ r,\ \mathsf{pr}_2\ r)$, then $\mathsf{transport}^P\ q\ (\mathsf{Hind}\ P\ \mathsf{pr}_2\ (s^{-1}\ y)) : P\ (\mathsf{pr}_1\ r,\ \mathsf{pr}_2\ r)$.

[18] In more detail, if we have proofs $t_1 : q = \mathsf{pr}_2\ n$ and $t_2 : P\ (\mathsf{pr}_1\ n, q)$, then $\mathsf{transport}^{(\lambda g.\ P\ (\mathsf{pr}_1\ n, g))}\ t_1\ t_2 : P\ (\mathsf{pr}_1\ n, \mathsf{pr}_2\ n)$.

The idea behind the induction principle for $\mathsf{A_I}$ is similar to the strategy followed by the extended Mendler-style induction principle (Corollary 2.4.4). To see the analogy, it is useful to interpret the induction principle in a set-theoretic language by identifying predicates (and their corresponding sigma types) with subsets, and regarding functor H as a map between sets.

For the set interpretation, we regard A_I as a subset of A, and T as an arbitrary subset of A. Also, unary predicate P is represented as a subset of A (i.e. the set of all elements in A satisfying the predicate).

Then, condition (3.10) can be written as:

$$\forall T.\ (T \subseteq A_I) \to (T \subseteq P) \to (In_{A_I}[H(T)] \subseteq P)$$

and the conclusion (3.11) can be written as:

$$A_I \subseteq P$$

where:

- $T \subseteq A_I$ denotes term $\mathsf{T} : \mathsf{A_I} \to \mathbf{Prop}_m$ (i.e. T is a subset of $A_I \subseteq A$).
- $T \subseteq P$ denotes type:

$$\prod_{\mathsf{w}:\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T\ y}} \mathsf{P}\ (\mathsf{pr_1\ w})$$

  i.e. "Any w in T, also satisfies P".
- $In_{A_I}[H(T)] \subseteq P$ denotes type:

$$\prod_{\mathsf{w}:\mathsf{H}\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T\ y}} \mathsf{P}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{A_I}}\ \mathsf{w})$$

  i.e. "For any w in $H(T)$ we have that $In_{A_I}(w)$ satisfies P", which is logically equivalent to: "Any w in $In_{A_I}[H(T)]$ satisfies P", where $In_{A_I}[H(T)]$ is the image of $H(T)$ under function In_{A_I}.

Notice that the set-theoretic translation looks a lot like Corollary 2.4.4, with $\subseteq$ acting as the order relation. Hence, the induction principle for $\mathsf{A_I}$ claims that every element of A_I satisfies P *if* every set in the following sequence satisfies P (see the explanation for Mendler-style induction after Corollary 2.4.3):

$$\emptyset\ ,\ G(\emptyset)\ ,\ G(G(\emptyset))\ ,\ G(G(G(\emptyset)))\ ,\ G(G(G(G(\emptyset))))\ ,\ \dots$$

where G is a function defined on subsets of A as $G(X) = In_{A_I}[H(X)]$.

Step 4: Prove initiality for $\mathsf{A_I}$

All previous steps worked with an arbitrary set H-algebra. For this step we require the stronger condition that the algebra is a LLWinSAlg or a WinSAlg. This means we will prove two cases, i.e. if A is a LLWinSAlg, then $\mathsf{A_I}$ is a LLInSAlg; if A is a WinSAlg, then $\mathsf{A_I}$ is an InSAlg.

Theorem 3.2.21. *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor.*

(i) Let $i > 0$ *and* $(\mathsf{A}, \mathsf{In_A}, \mathsf{m}) : \mathsf{LLWinSAlg}_i\ \mathsf{H}$ *a lower level weakly initial set* H*-algebra. Then the tuple* $(\mathsf{A_I}, \mathsf{In_{A_I}}, \mathsf{mc})$ *is a lower level initial set* H*-algebra, where* mc *is a proof that the type of morphisms going out of* $\mathsf{A_I}$ *is contractible.*

In other words, we have the tuple:[19]

$$(\mathsf{A_I}, \mathsf{In_{A_I}}, \mathsf{mc}) : \mathsf{LLInSAlg}_i\ \mathsf{H}$$

(ii) Let $(\mathsf{A}, \mathsf{In_A}, \mathsf{m}) : \mathsf{WinSAlg}_i\ \mathsf{H}$ *be a weakly initial set* H*-algebra. Then the tuple* $(\mathsf{A_I}, \mathsf{In_{A_I}}, \mathsf{mc})$ *is an initial set* H*-algebra, where* mc *is a proof that the type of morphisms going out of* $\mathsf{A_I}$ *is contractible.*

In other words, we have the tuple:

$$(\mathsf{A_I}, \mathsf{In_{A_I}}, \mathsf{mc}) : \mathsf{InSAlg}_i\ \mathsf{H}$$

[19] By Lemma 3.2.18, we have that $\mathsf{A_I}$ is at universe level i when algebra A is at level $k \leq i$. In this case $k = i$.

Proof. The proof for both results is almost identical. As such, we will develop the proof for (i). The proof for (ii) is obtained by replacing all occurrences of universe level $i-1$ for universe level i.

We have to construct a proof:

$$\mathsf{mc}: \prod_{(\mathsf{B},\mathsf{In_B}):\mathsf{SAlg}_{i-1}\ \mathsf{H}} \mathsf{IsContr}\ (\mathsf{AlgMor}\ \mathsf{A_I}\ \mathsf{B})$$

Let $(\mathsf{B},\mathsf{In_B}):\mathsf{SAlg}_{i-1}\ \mathsf{H}$.

We have to build a term of type $\mathsf{IsContr}\ (\mathsf{AlgMor}\ \mathsf{A_I}\ \mathsf{B})$. By definition of contractible type, this means we have to build a term of type:

$$\sum_{(\mathsf{h},\mathsf{w_1}):\mathsf{AlgMor}\ \mathsf{A_I}\ \mathsf{B}} \prod_{(\mathsf{g},\mathsf{w_2}):\mathsf{AlgMor}\ \mathsf{A_I}\ \mathsf{B}} (\mathsf{h},\mathsf{w_1}) = (\mathsf{g},\mathsf{w_2})$$

Intuitively, this means we have to prove the "existence" of a morphism $(\mathsf{h},\mathsf{w_1})$ from $\mathsf{A_I}$ to B, and the "uniqueness" of such morphism.

Existence

By hypothesis, we have a term:

$$\mathsf{m}: \prod_{(\mathsf{C},\mathsf{In_C}):\mathsf{SAlg}_{i-1}\ \mathsf{H}} \mathsf{AlgMor}\ \mathsf{A}\ \mathsf{C}$$

This means we have a function:

$$\mathsf{pr_1}\ (\mathsf{m}\ (\mathsf{B},\mathsf{In_B})):\mathsf{A}\to\mathsf{B}$$

and also $\mathsf{pr_2}\ (\mathsf{m}\ (\mathsf{B},\mathsf{In_B}))$ is a proof that this function is a morphism.

Therefore, we can define h as the composition:

$$\mathsf{A_I} \xrightarrow{\mathsf{pr_1}} \mathsf{A} \xrightarrow{\mathsf{pr_1}\ (\mathsf{m}\ (\mathsf{B},\mathsf{In_B}))} \mathsf{B}$$

It remains to build a proof $\mathsf{w_1}$ that h is a morphism. But h is the composition of two morphisms, since $\mathsf{pr_1}$ is a morphism by Lemma 3.2.19, and $\mathsf{pr_2}\ (\mathsf{m}\ (\mathsf{B},\mathsf{In_B}))$ is a proof that $\mathsf{pr_1}\ (\mathsf{m}\ (\mathsf{B},\mathsf{In_B}))$ is a morphism. Therefore, by Lemma 1.9.13, h is a morphism.[20] This means we have the commutative diagram:

$$\begin{array}{ccc} \mathsf{H}\ \mathsf{A_I} & \xrightarrow{\mathsf{map^H}\ \mathsf{h}} & \mathsf{H}\ \mathsf{B} \\ {\scriptstyle \mathsf{In_{A_I}}}\downarrow & & \downarrow{\scriptstyle \mathsf{In_B}} \\ \mathsf{A_I} & \xrightarrow[\mathsf{h}]{} & \mathsf{B} \end{array} \tag{3.13}$$

Uniqueness

We have to show that the following type is inhabited:

$$\prod_{(\mathsf{g},\mathsf{w_2}):\mathsf{AlgMor}\ \mathsf{A_I}\ \mathsf{B}} (\mathsf{h},\mathsf{w_1}) = (\mathsf{g},\mathsf{w_2})$$

where $(\mathsf{h},\mathsf{w_1})$ is the morphism built in the existence part.

Let $(\mathsf{g},\mathsf{w_2}):\mathsf{AlgMor}\ \mathsf{A_I}\ \mathsf{B}$. In other words, we have the commutative diagram:

$$\begin{array}{ccc} \mathsf{H}\ \mathsf{A_I} & \xrightarrow{\mathsf{map^H}\ \mathsf{g}} & \mathsf{H}\ \mathsf{B} \\ {\scriptstyle \mathsf{In_{A_I}}}\downarrow & & \downarrow{\scriptstyle \mathsf{In_B}} \\ \mathsf{A_I} & \xrightarrow[\mathsf{g}]{} & \mathsf{B} \end{array} \tag{3.14}$$

[20] We are only interested in the existence of a proof $\mathsf{w_1}$, we do not require the specific term $\mathsf{w_1}$ stands for.

We know that type $\mathsf{AlgMor}\ \mathsf{A_I}\ \mathsf{B}$ is defined as (see Definition 1.9.12):

$$\sum_{\mathsf{f}:\mathsf{A_I}\to\mathsf{B}} \prod_{\mathsf{w}:\mathsf{H}\ \mathsf{A_I}} \mathsf{f}\ (\mathsf{In}_{\mathsf{A_I}}\ \mathsf{w}) = \mathsf{In}_{\mathsf{B}}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{f}\ \mathsf{w})$$

The second component of this sigma type, i.e.,

$$\prod_{\mathsf{w}:\mathsf{H}\ \mathsf{A_I}} \mathsf{f}\ (\mathsf{In}_{\mathsf{A_I}}\ \mathsf{w}) = \mathsf{In}_{\mathsf{B}}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{f}\ \mathsf{w})$$

is a mere proposition for every $\mathsf{f}\colon \mathsf{A_I} \to \mathsf{B}$, since both terms appearing in the equality are of type B (which is a set), and propositions are closed under Π-types (see Lemma 1.8.17).

Therefore, by Lemma 1.8.14, to prove that $(\mathsf{h}, \mathsf{w}_1) = (\mathsf{g}, \mathsf{w}_2)$ is inhabited, it is enough to prove that $\mathsf{h} = \mathsf{g}$ holds. However, by function extensionality (see Section 1.6.1), it is enough to prove:

$$\prod_{\mathsf{w}:\mathsf{A_I}} \mathsf{h}\ \mathsf{w} = \mathsf{g}\ \mathsf{w}$$

To prove this, we will use the induction principle for $\mathsf{A_I}$ (Lemma 3.2.20) instantiated with the predicate:

$$\mathsf{P} :\equiv (\lambda \mathsf{w}:\mathsf{A_I}.\ \mathsf{h}\ \mathsf{w} = \mathsf{g}\ \mathsf{w})\colon \mathsf{A_I} \to \mathbf{Prop}_{i-1}$$

In other words, if we can prove that the following type is inhabited:

$$\prod_{\mathsf{T}:\mathsf{A_I}\to\mathbf{Prop}_{i-1}} \left(\prod_{\mathsf{w}:\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y}} \mathsf{P}\ (\mathsf{pr}_1\ \mathsf{w}) \right) \to \left(\prod_{\mathsf{w}:\mathsf{H}\ \sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y}} \mathsf{P}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{A_I}}\ \mathsf{w}) \right)$$

then, the following type will be inhabited:

$$\prod_{\mathsf{w}:\mathsf{A_I}} \mathsf{h}\ \mathsf{w} = \mathsf{g}\ \mathsf{w}$$

Let $\mathsf{T}\colon \mathsf{A_I} \to \mathbf{Prop}_{i-1}$ such that the following type is inhabited:

$$\prod_{\mathsf{w}:\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y}} \mathsf{P}\ (\mathsf{pr}_1\ \mathsf{w})$$

or equivalently, by definition of P:

$$\prod_{\mathsf{w}:\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y}} \mathsf{h}\ (\mathsf{pr}_1\ \mathsf{w}) = \mathsf{g}\ (\mathsf{pr}_1\ \mathsf{w}) \tag{3.15}$$

We have to prove that the following type is inhabited:

$$\prod_{\mathsf{w}:\mathsf{H}\ \sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y}} \mathsf{P}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{A_I}}\ \mathsf{w})$$

or equivalently, by definition of P:

$$\prod_{\mathsf{w}:\mathsf{H}\ \sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y}} \mathsf{h}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{A_I}}\ \mathsf{w}) = \mathsf{g}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{A_I}}\ \mathsf{w})$$

Define the function:

$$\mathsf{ts} :\equiv \Sigma_{\mathsf{map}}\ \mathsf{id}_{\mathsf{A_I}}\ (\lambda(\mathsf{w}:\mathsf{A_I})(\mathsf{y}:\mathsf{T}\ \mathsf{w}).\ \star)$$

which has type:

$$\left(\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y} \right) \xrightarrow{\mathsf{ts}} \left(\sum_{\mathsf{y}:\mathsf{A_I}} \mathbb{1} \right)$$

Now, we prove a claim.

Claim 1: The following diagram commutes:

$$\left(\mathsf{H} \sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y}\right) \xrightarrow{\mathsf{map^H}\ \mathsf{ts}} \left(\mathsf{H} \sum_{\mathsf{y}:\mathsf{A_I}} \mathbb{1}\right) \xrightarrow{\mathsf{map^H}\ \mathsf{pr_1}} \mathsf{H}\ \mathsf{A_I} \underset{\mathsf{map^H}\ \mathsf{g}}{\overset{\mathsf{map^H}\ \mathsf{h}}{\rightrightarrows}} \mathsf{H}\ \mathsf{B} \tag{3.16}$$

First, we prove that the following diagram commutes:

$$\left(\sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y}\right) \xrightarrow{\mathsf{ts}} \left(\sum_{\mathsf{y}:\mathsf{A_I}} \mathbb{1}\right) \xrightarrow{\mathsf{pr_1}} \mathsf{A_I} \underset{\mathsf{g}}{\overset{\mathsf{h}}{\rightrightarrows}} \mathsf{B} \tag{3.17}$$

Let $(\mathsf{a}, \mathsf{b}) : \sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y}$. Then, we have:

$$\begin{aligned}
\mathsf{h}\ (\mathsf{pr_1}\ (\mathsf{ts}\ (\mathsf{a}, \mathsf{b}))) &\overset{(1)}{=} \mathsf{h}\ (\mathsf{pr_1}\ (\mathsf{a}, \star)) \\
&\overset{(2)}{=} \mathsf{h}\ \mathsf{a} \\
&\overset{(3)}{=} \mathsf{g}\ \mathsf{a} \\
&\overset{(4)}{=} \mathsf{g}\ (\mathsf{pr_1}\ (\mathsf{a}, \star)) \\
&\overset{(5)}{=} \mathsf{g}\ (\mathsf{pr_1}\ (\mathsf{ts}\ (\mathsf{a}, \mathsf{b})))
\end{aligned}$$

where (1) and (5) follow by definition of ts and the Σ_{map} function (see Lemma 1.5.28), (2) and (4) by definition of $\mathsf{pr_1}$, and (3) by the inductive hypothesis (type (3.15)).[21]

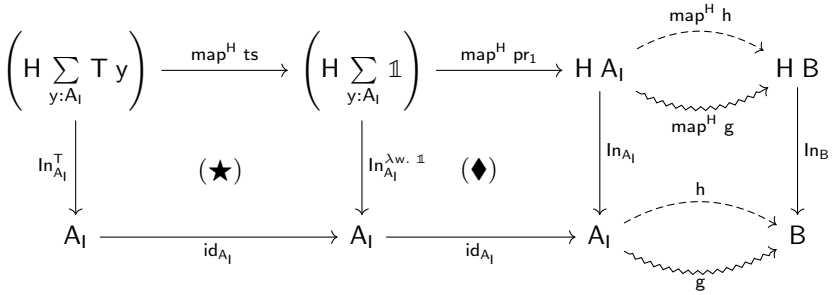
Now, we are ready to prove the claim. Let $\mathsf{w} : \mathsf{H} \sum_{\mathsf{y}:\mathsf{A_I}} \mathsf{T}\ \mathsf{y}$. Then, we have:

$$\begin{aligned}
\mathsf{map^H}\ \mathsf{h}\ (\mathsf{map^H}\ \mathsf{pr_1}\ (\mathsf{map^H}\ \mathsf{ts}\ \mathsf{w})) &\overset{(1)}{=} \mathsf{map^H}\ (\mathsf{h} \circ \mathsf{pr_1} \circ \mathsf{ts})\ \mathsf{w} \\
&\overset{(2)}{=} \mathsf{map^H}\ (\mathsf{g} \circ \mathsf{pr_1} \circ \mathsf{ts})\ \mathsf{w} \\
&\overset{(3)}{=} \mathsf{map^H}\ \mathsf{g}\ (\mathsf{map^H}\ \mathsf{pr_1}\ (\mathsf{map^H}\ \mathsf{ts}\ \mathsf{w}))
\end{aligned}$$

where (1) and (3) follow by functorial mapping of H (see Definition 1.9.6), and (2) follows by Diagram (3.17) and function extensionality.

This proves the claim.

Now, we have the following diagram:



(3.18)

where the square marked with ($\bigstar$) follows by Lemma 3.2.13 since the identity function is a morphism (Lemma 1.9.14), the square marked with ($\blacklozenge$) follows by Lemma 3.2.14, the square using dashed arrows[22] ($\dashrightarrow$) is Diagram (3.13), and the square using squiggly arrows[23] ($\rightsquigarrow$) is Diagram (3.14). Each one of these four squares commutes.

[21] In more detail, we have a term q having type (3.15), therefore $\mathsf{q}\ (\mathsf{a}, \mathsf{b}) : \mathsf{h}\ (\mathsf{pr_1}\ (\mathsf{a}, \mathsf{b})) = \mathsf{g}\ (\mathsf{pr_1}\ (\mathsf{a}, \mathsf{b}))$ which is equivalent to $\mathsf{q}\ (\mathsf{a}, \mathsf{b}) : \mathsf{h}\ \mathsf{a} = \mathsf{g}\ \mathsf{a}$ by definition of the projection function $\mathsf{pr_1}$.

[22] The square includes functions $\mathsf{In}_{\mathsf{A_I}}$ and In_{B}.

[23] The square includes functions $\mathsf{In}_{\mathsf{A_I}}$ and In_{B}.

We are trying to prove that the following type is inhabited:

$$\prod_{w:H \sum_{y:A_I} T\ y} h\ (\mathsf{In}^T_{A_I}\ w) = g\ (\mathsf{In}^T_{A_I}\ w)$$

Let $w : H \sum_{y:A_I} T\ y$. By chasing Diagram (3.18), we have the following sequence of identifications:[24]

$$\begin{aligned}
h\ (\mathsf{In}^T_{A_I}\ w) &\overset{(1)}{=} h\ (\mathsf{id}_{A_I}\ (\mathsf{id}_{A_I}\ (\mathsf{In}^T_{A_I}\ w))) \\
&\overset{(2)}{=} h\ (\mathsf{id}_{A_I}\ (\mathsf{In}^{\lambda w.\ \mathbb{1}}_{A_I}\ (\mathsf{map}^H\ ts\ w))) \\
&\overset{(3)}{=} h\ (\mathsf{In}_{A_I}\ (\mathsf{map}^H\ \mathsf{pr}_1\ (\mathsf{map}^H\ ts\ w))) \\
&\overset{(4)}{=} \mathsf{In}_B\ (\mathsf{map}^H\ h\ (\mathsf{map}^H\ \mathsf{pr}_1\ (\mathsf{map}^H\ ts\ w))) \\
&\overset{(5)}{=} \mathsf{In}_B\ (\mathsf{map}^H\ g\ (\mathsf{map}^H\ \mathsf{pr}_1\ (\mathsf{map}^H\ ts\ w))) \\
&\overset{(6)}{=} g\ (\mathsf{In}_{A_I}\ (\mathsf{map}^H\ \mathsf{pr}_1\ (\mathsf{map}^H\ ts\ w))) \\
&\overset{(7)}{=} g\ (\mathsf{id}_{A_I}\ (\mathsf{In}^{\lambda w.\ \mathbb{1}}_{A_I}\ (\mathsf{map}^H\ ts\ w))) \\
&\overset{(8)}{=} g\ (\mathsf{id}_{A_I}\ (\mathsf{id}_{A_I}\ (\mathsf{In}^T_{A_I}\ w))) \\
&\overset{(9)}{=} g\ (\mathsf{In}^T_{A_I}\ w)
\end{aligned}$$

where (1) and (9) follow by definition of the identity function id_{A_I}, (2) and (8) by commutativity of the square (★), (3) and (7) by commutativity of the square (⧫), (4) by commutativity of the square with dashed arrows (⇢) or Diagram (3.13), (5) by commutativity of Diagram (3.16) in Claim 1, and (6) by commutativity of the square with squiggly arrows (⇝) or Diagram (3.14).

□

Theorem 3.2.21 states that the problem of building a (lower level) initial set algebra is reduced to the problem of building a (lower level) weakly initial set algebra.

The theorem has the following corollary.

Corollary 3.2.22. *Let $i > 0$, and $H : \mathcal{U}_i \to \mathcal{U}_i$ a functor. Then, H has a lower level initial set H-algebra. In other words, the following type is inhabited:*

$$\mathsf{LLInSAlg}_i\ H$$

Proof. By Lemma 3.2.7, H has a lower level weakly initial set H-algebra at universe level i. Denote this LLWinSAlg by (W, In_W, m).

But then, by Theorem 3.2.21(i), $(W_I, \mathsf{In}_{W_I}, mc)$ is a lower level initial set H-algebra at universe level i.

□

Therefore, any endofunctor has a *lower level* initial set H-algebra. What about an initial set H-algebra?

3.2.3 A functor without an initial set algebra

In set-based mathematics, it is well-known that the powerset functor does not have an initial algebra (see Section 10.5 in [5]). The powerset functor maps each set X into its powerset $\mathcal{P}(X)$, and each function $f : A \to B$ into a function $g : \mathcal{P}(A) \to \mathcal{P}(B)$ such that for each $X \in \mathcal{P}(A)$, we have $g(X) = f[X]$ (i.e. g maps X into the image of X under function f).

We can reproduce this classic result in HoTT. First, we need to represent the powerset functor. However, we will call it "powertype" functor, because it is defined for arbitrary types.

[24] These identifications make implicit use of the ap operator almost everywhere (see Lemma 1.5.62). For example, in step (2), by commutativity of the square (★), there is a proof $q : \mathsf{id}_{A_I}\ (\mathsf{In}^T_{A_I}\ w) = \mathsf{In}^{\lambda w.\ \mathbb{1}}_{A_I}\ (\mathsf{map}^H\ ts\ w)$.

Therefore, $\mathsf{ap}_h\ (\mathsf{ap}_{\mathsf{id}_{A_I}}\ q) : h\ (\mathsf{id}_{A_I}\ (\mathsf{id}_{A_I}\ (\mathsf{In}^T_{A_I}\ w))) = h\ (\mathsf{id}_{A_I}\ (\mathsf{In}^{\lambda w.\ \mathbb{1}}_{A_I}\ (\mathsf{map}^H\ ts\ w)))$ by the ap operator applied twice.

We omit the details for the other steps regarding the use of the ap operator, since the pattern is similar.

Definition 3.2.23 (Powertype functor)**.** Let $i > 0$ and $j < i$ two universe levels. We define the function $\mathsf{Pow}_{i,j} : \mathcal{U}_i \to \mathcal{U}_i$ as:

$$\mathsf{Pow}_{i,j} :\equiv (\lambda \mathsf{Y} : \mathcal{U}_i.\ \mathsf{Y} \to \mathbf{Prop}_j) : \mathcal{U}_i \to \mathcal{U}_i$$

and a mapping function:

$$\mathsf{map}^{\mathsf{Pow}_{i,j}} :\equiv \lambda(\mathsf{A}:\mathcal{U}_i)(\mathsf{B}:\mathcal{U}_i)(\mathsf{f}:\mathsf{A}\to\mathsf{B})(\mathsf{P}:\mathsf{A}\to\mathbf{Prop}_j).\ \lambda \mathsf{w}:\mathsf{B}.\ (\mathsf{GPropR}^i_j)^{-1}\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{f}\ \mathsf{y} = \mathsf{w})\right\|$$

having type:

$$\prod_{\mathsf{A},\mathsf{B}:\mathcal{U}_i} (\mathsf{A}\to\mathsf{B}) \to ((\mathsf{A}\to\mathbf{Prop}_j) \to (\mathsf{B}\to\mathbf{Prop}_j))$$

▲

First, we explain that functions $\mathsf{Pow}_{i,j}$ and $\mathsf{map}^{\mathsf{Pow}_{i,j}}$ are well-typed. Type $\mathsf{Y} \to \mathbf{Prop}_j$ is at level $\max\{i, j+1\} = i$, since $\mathbf{Prop}_j : \mathcal{U}_{j+1}$ and $j+1 \leq i$. Also, the sigma type in $\mathsf{map}^{\mathsf{Pow}_{i,j}}$ is (-1)-truncated because it is required to be a proposition (i.e. A is an arbitrary type, which means that the sigma type is not necessarily a proposition). Function $\mathsf{map}^{\mathsf{Pow}_{i,j}}$ requires the use of the propositional resizing equivalence $\mathsf{GPropR}^i_j : \mathbf{Prop}_j \to \mathbf{Prop}_i$ (see Lemma 1.8.22), since type:

$$\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{f}\ \mathsf{y} = \mathsf{w})\right\|$$

is at level $\max\{i, j, i\} = i$, but it is required to be at level j.

The definition of $\mathsf{Pow}_{i,j}$ is imitating the classical definition of the powerset functor. In HoTT, the "powertype" of a type is the type of all predicates on it (Definition 1.8.19). Therefore, $\mathsf{Pow}_{i,j}$ maps a type Y into its powertype $\mathsf{Y} \to \mathbf{Prop}_j$. Similarly, the mapping function $\mathsf{map}^{\mathsf{Pow}_{i,j}}$ takes a function $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ and a subtype $\mathsf{P} : \mathsf{A} \to \mathbf{Prop}_j$ of A, and returns the "image" of P under function f. To see that predicate:

$$\lambda \mathsf{w}:\mathsf{B}.\ (\mathsf{GPropR}^i_j)^{-1}\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{f}\ \mathsf{y} = \mathsf{w})\right\|$$

is the "image" of P under f, it is useful to recast it in set-theoretic language, as follows:[25]

$$\{x \in B \mid \exists y \in A, (y \in P) \wedge (f(y) = x)\} = \{x \in B \mid \exists y \in P, f(y) = x\} = f[P]$$

where $P \subseteq A$ (remember that predicates can be interpreted as subsets, and (-1)-truncated sigmas as classical existentials, see Remark 1.11.5).

Now, function $\mathsf{Pow}_{i,j}$ and its mapping function $\mathsf{map}^{\mathsf{Pow}_{i,j}}$ *form* a functor.

Lemma 3.2.24. *Let $i > 0$ and $j < i$ be two universe levels. Function $\mathsf{Pow}_{i,j}$ (together with its mapping function $\mathsf{map}^{\mathsf{Pow}_{i,j}}$) is an endofunctor that preserves sets.*

Proof. It is obvious from its definition that $\mathsf{Pow}_{i,j}$ is an endofunctor. Also, for any type Y we have that $\mathsf{Y} \to \mathbf{Prop}_j$ is a set, because $\mathbf{Prop}_j$ is a set by Lemma 1.8.18, and sets are closed under Π-types (Lemma 1.8.18). Hence, $\mathsf{Pow}_{i,j}$ preserves sets.

Now, we need to show that $\mathsf{Pow}_{i,j}$ satisfies the two functorial properties (see Definition 1.9.6).

- Type $\prod_{\mathsf{A}:\mathcal{U}_i} \prod_{\mathsf{P}:\mathsf{A}\to\mathbf{Prop}_j} \mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{P} = \mathsf{P}$ is inhabited.

  Let $\mathsf{A} : \mathcal{U}_i$ and $\mathsf{P} : \mathsf{A} \to \mathbf{Prop}_j$. By function extensionality. it is enough to prove that the following type is inhabited:

  $$\prod_{\mathsf{w}:\mathsf{A}} \mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{P}\ \mathsf{w} = \mathsf{P}\ \mathsf{w}$$

[25] Although we are dealing with general types and not sets, it is illuminating to interpret this predicate *as if* it were using sets, because doing such thing helps in understanding the definition.

Let $\mathsf{w} : \mathsf{A}$. We are trying to prove an equality between propositions. Therefore, by univalence, it is enough to prove that they are equivalent types. But, by Lemma 1.8.8, it is enough to prove that these propositions are *logically equivalent*. In other words, we need to prove that the following type is inhabited:

$$((\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{P}\ \mathsf{w}) \to (\mathsf{P}\ \mathsf{w})) \times ((\mathsf{P}\ \mathsf{w}) \to (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{P}\ \mathsf{w}))$$

First, we prove $(\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{P}\ \mathsf{w}) \to (\mathsf{P}\ \mathsf{w})$, or equivalently:

$$\left((\mathsf{GPropR}_j^i)^{-1}\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{id}_{\mathsf{A}}\ \mathsf{y} = \mathsf{w})\right\|\right) \to (\mathsf{P}\ \mathsf{w})$$

By Lemma 1.8.24, it is enough to prove:

$$\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{id}_{\mathsf{A}}\ \mathsf{y} = \mathsf{w})\right\| \to (\mathsf{P}\ \mathsf{w})$$

By (-1)-truncation recursion (since $\mathsf{P}\ \mathsf{w}$ is a proposition), it is enough to prove (after simplification):

$$\left(\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{y} = \mathsf{w})\right) \to (\mathsf{P}\ \mathsf{w})$$

Let $(\mathsf{y}, \mathsf{q}, \mathsf{r}) : \sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{y} = \mathsf{w})$. Then, we have $\mathsf{transport}^{\mathsf{P}}\ \mathsf{r}\ \mathsf{q} : \mathsf{P}\ \mathsf{w}$, as required.

For the other direction, we have to prove $(\mathsf{P}\ \mathsf{w}) \to (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{P}\ \mathsf{w})$, or equivalently (after simplification):

$$(\mathsf{P}\ \mathsf{w}) \to \left((\mathsf{GPropR}_j^i)^{-1}\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{y} = \mathsf{w})\right\|\right)$$

Let $\mathsf{q} : \mathsf{P}\ \mathsf{w}$. We need to construct a term of type:

$$(\mathsf{GPropR}_j^i)^{-1}\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{y} = \mathsf{w})\right\|$$

By Lemma 1.8.23, it is enough to construct a term of type:

$$\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{y} = \mathsf{w})\right\|$$

But then, we have:

$$|(\mathsf{w}, \mathsf{q}, \mathsf{refl}_{\mathsf{A}}\ \mathsf{w})| : \left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{y} = \mathsf{w})\right\|$$

- The following type is inhabited:

$$\prod_{\mathsf{A},\mathsf{B},\mathsf{C}:\mathcal{U}_i}\ \prod_{\mathsf{g}:\mathsf{A}\to\mathsf{B}}\ \prod_{\mathsf{h}:\mathsf{B}\to\mathsf{C}}\ \prod_{\mathsf{P}:\mathsf{A}\to\mathbf{Prop}_j} \mathsf{map}^{\mathsf{Pow}_{i,j}}\ (\mathsf{h} \circ \mathsf{g})\ \mathsf{P} = \mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{h}\ (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P})$$

Let $\mathsf{A}, \mathsf{B}, \mathsf{C} : \mathcal{U}_i$, $\mathsf{g} : \mathsf{A} \to \mathsf{B}$, $\mathsf{h} : \mathsf{B} \to \mathsf{C}$, and $\mathsf{P} : \mathsf{A} \to \mathbf{Prop}_j$. By function extensionality, it is enough to prove:

$$\prod_{\mathsf{w}:\mathsf{C}} \mathsf{map}^{\mathsf{Pow}_{i,j}}\ (\mathsf{h} \circ \mathsf{g})\ \mathsf{P}\ \mathsf{w} = \mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{h}\ (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P})\ \mathsf{w}$$

Let $\mathsf{w}\colon \mathsf{C}$. We are trying to prove an equality between propositions. Therefore, by univalence and Lemma 1.8.8, it is enough to prove that they are *logically equivalent*.

First, we prove $(\mathsf{map}^{\mathsf{Pow}_{i,j}}\ (\mathsf{h} \circ \mathsf{g})\ \mathsf{P}\ \mathsf{w}) \to (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{h}\ (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P})\ \mathsf{w})$, or equivalently:

$$\left((\mathsf{GPropR}^i_j)^{-1}\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{h}\ (\mathsf{g}\ \mathsf{y}) = \mathsf{w})\right\|\right) \to \left((\mathsf{GPropR}^i_j)^{-1}\left\|\sum_{\mathsf{z}:\mathsf{B}} (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P}\ \mathsf{z}) \times (\mathsf{h}\ \mathsf{z} = \mathsf{w})\right\|\right)$$

By Lemma 1.8.24 and (-1)-truncation recursion, it is enough to prove:

$$\left(\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{h}\ (\mathsf{g}\ \mathsf{y}) = \mathsf{w})\right) \to \left((\mathsf{GPropR}^i_j)^{-1}\left\|\sum_{\mathsf{z}:\mathsf{B}} (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P}\ \mathsf{z}) \times (\mathsf{h}\ \mathsf{z} = \mathsf{w})\right\|\right)$$

Let $(\mathsf{y}, \mathsf{q}, \mathsf{r})\colon \sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{h}\ (\mathsf{g}\ \mathsf{y}) = \mathsf{w})$. By Lemma 1.8.23, it is enough to construct a term of type:

$$\left\|\sum_{\mathsf{z}:\mathsf{B}} (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P}\ \mathsf{z}) \times (\mathsf{h}\ \mathsf{z} = \mathsf{w})\right\|$$

If we are able to construct a term $\mathsf{t}\colon \mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P}\ (\mathsf{g}\ \mathsf{y})$, we will have:

$$|(\mathsf{g}\ \mathsf{y}, \mathsf{t}, \mathsf{r})|\colon \left\|\sum_{\mathsf{z}:\mathsf{B}} (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P}\ \mathsf{z}) \times (\mathsf{h}\ \mathsf{z} = \mathsf{w})\right\|$$

Therefore, it remains to prove that type $\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P}\ (\mathsf{g}\ \mathsf{y})$ is inhabited, or equivalently:

$$(\mathsf{GPropR}^i_j)^{-1}\left\|\sum_{\mathsf{z}:\mathsf{A}} (\mathsf{P}\ \mathsf{z}) \times (\mathsf{g}\ \mathsf{z} = \mathsf{g}\ \mathsf{y})\right\|$$

By Lemma 1.8.23 again, it is enough to construct a term of type:

$$\left\|\sum_{\mathsf{z}:\mathsf{A}} (\mathsf{P}\ \mathsf{z}) \times (\mathsf{g}\ \mathsf{z} = \mathsf{g}\ \mathsf{y})\right\|$$

But we have:

$$|(\mathsf{y}, \mathsf{q}, \mathsf{refl}_{\mathsf{B}}\ (\mathsf{g}\ \mathsf{y}))|\colon \left\|\sum_{\mathsf{z}:\mathsf{A}} (\mathsf{P}\ \mathsf{z}) \times (\mathsf{g}\ \mathsf{z} = \mathsf{g}\ \mathsf{y})\right\|$$

For the other direction, we now prove:

$$(\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{h}\ (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P})\ \mathsf{w}) \to (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ (\mathsf{h} \circ \mathsf{g})\ \mathsf{P}\ \mathsf{w})$$

or equivalently:

$$\left((\mathsf{GPropR}^i_j)^{-1}\left\|\sum_{\mathsf{z}:\mathsf{B}} (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P}\ \mathsf{z}) \times (\mathsf{h}\ \mathsf{z} = \mathsf{w})\right\|\right) \to \left((\mathsf{GPropR}^i_j)^{-1}\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{h}\ (\mathsf{g}\ \mathsf{y}) = \mathsf{w})\right\|\right)$$

By Lemma 1.8.24 and (-1)-truncation recursion, it is enough to prove:

$$\left(\sum_{\mathsf{z}:\mathsf{B}} (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P}\ \mathsf{z}) \times (\mathsf{h}\ \mathsf{z} = \mathsf{w})\right) \to \left((\mathsf{GPropR}^i_j)^{-1}\left\|\sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{h}\ (\mathsf{g}\ \mathsf{y}) = \mathsf{w})\right\|\right)$$

Let $(\mathsf{z}, \mathsf{q}, \mathsf{r}) : \sum_{\mathsf{z}:\mathsf{B}} (\mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P}\ \mathsf{z}) \times (\mathsf{h}\ \mathsf{z} = \mathsf{w})$.

By Lemma 1.8.23, it is enough to construct a term of type:

$$\left\| \sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{h}\ (\mathsf{g}\ \mathsf{y}) = \mathsf{w}) \right\|$$

Now, by hypothesis we have a term $\mathsf{q} : \mathsf{map}^{\mathsf{Pow}_{i,j}}\ \mathsf{g}\ \mathsf{P}\ \mathsf{z}$, or equivalently, we have:

$$\mathsf{q} : (\mathsf{GPropR}^i_j)^{-1} \left\| \sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{g}\ \mathsf{y} = \mathsf{z}) \right\|$$

Hence, by Lemma 1.8.23, there is a term:

$$\mathsf{q}_1 : \left\| \sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{g}\ \mathsf{y} = \mathsf{z}) \right\|$$

Then, by (-1)-truncation recursion, there will be a term:

$$(\mathsf{t}, \mathsf{s}, \mathsf{u}) : \sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{g}\ \mathsf{y} = \mathsf{z})$$

Therefore, we can construct a proof:[26]

$$(\mathsf{ap}_{\mathsf{h}}\ \mathsf{u}) \centerdot \mathsf{r} : \mathsf{h}\ (\mathsf{g}\ \mathsf{t}) = \mathsf{w}$$

Which means:

$$|(\mathsf{t}, \mathsf{s}, (\mathsf{ap}_{\mathsf{h}}\ \mathsf{u}) \centerdot \mathsf{r})| : \left\| \sum_{\mathsf{y}:\mathsf{A}} (\mathsf{P}\ \mathsf{y}) \times (\mathsf{h}\ (\mathsf{g}\ \mathsf{y}) = \mathsf{w}) \right\|$$

□

By Corollary 3.2.22, the $\mathsf{Pow}_{i,j}$ functor has a lower level initial set $\mathsf{Pow}_{i,j}$-algebra. However, it does not have an initial set $\mathsf{Pow}_{i,j}$-algebra, as the following theorem shows.

Theorem 3.2.25. *Let $i > 0$ and $j < i$ be two universe levels. The functor $\mathsf{Pow}_{i,j}$ does not have an initial set $\mathsf{Pow}_{i,j}$-algebra. In other words, the following type is inhabited:*

$$\neg(\mathsf{InSAlg}_i\ \mathsf{Pow}_{i,j})$$

or, equivalently, type $\mathsf{InSAlg}_i\ \mathsf{Pow}_{i,j}$ is not *inhabited.*

Proof. Suppose $(\mathsf{A}, \mathsf{in}_{\mathsf{A}}, \mathsf{mc}) : \mathsf{InSAlg}_i\ \mathsf{Pow}_{i,j}$ is an initial set $\mathsf{Pow}_{i,j}$-algebra. We want to prove that this leads to a contradiction (i.e. type $\mathbb{0}$ is inhabited).

Since $\mathsf{Pow}_{i,j}$ is a set-preserving endofunctor (Lemma 3.2.24), we can apply Lambek's Lemma (Lemma 3.2.11) to obtain that $\mathsf{Pow}_{i,j}\ \mathsf{A}$ and A are equivalent types.

Therefore, by definition of $\mathsf{Pow}_{i,j}$, types $\mathsf{A} \to \mathbf{Prop}_j$ and A are equivalent, which is impossible by Lemma 1.8.20 (i.e. by Lemma 1.8.20, type $\mathbb{0}$ is inhabited because $(\mathsf{A} \to \mathbf{Prop}_j) \simeq \mathsf{A}$ is inhabited).

□

Theorem 3.2.25 implies that any LLInSAlg for functor $\mathsf{Pow}_{i,j}$ is not an InSAlg. In particular, the LLInSAlg constructed by Corollary 3.2.22, is not an InSAlg for functor $\mathsf{Pow}_{i,j}$. Hence, not every LLInSAlg is an InSAlg.

We also have a corollary regarding the non-existence of weakly initial set algebras for functor $\mathsf{Pow}_{i,j}$.

[26] See Lemma 1.5.60 for the transitivity operator $(\centerdot)$.

Corollary 3.2.26. *Let $i > 0$ and $j < i$ be two universe levels. The functor $\mathsf{Pow}_{i,j}$ does not have a weakly initial set $\mathsf{Pow}_{i,j}$-algebra. In other words, the following type is inhabited:*

$$\neg(\mathsf{WinSAlg}_i\ \mathsf{Pow}_{i,j})$$

or, equivalently, type $\mathsf{WinSAlg}_i\ \mathsf{Pow}_{i,j}$ is not *inhabited.*

Proof. Suppose $(\mathsf{A}, \mathsf{In}_\mathsf{A}, \mathsf{m}) : \mathsf{WinSAlg}_i\ \mathsf{Pow}_{i,j}$.

By Theorem 3.2.21(ii), $(\mathsf{A}_\mathsf{I}, \mathsf{In}_{\mathsf{A}_\mathsf{I}}, \mathsf{mc})$ is an initial set $\mathsf{Pow}_{i,j}$-algebra, which contradicts Theorem 3.2.25. □

Another corollary states that no LLInSAlg for $\mathsf{Pow}_{i,j}$ satisfies the conclusion of Lambek's Lemma. In other words, a LLInSAlg will not be, in general, a "fixpoint" for its functor.

Corollary 3.2.27. *Let $i > 0$ and $j < i$ be two universe levels, and $(\mathsf{A}, \mathsf{In}_\mathsf{A}, \mathsf{mc})$ a lower level initial set $\mathsf{Pow}_{i,j}$-algebra. Then, $\mathsf{Pow}_{i,j}\ \mathsf{A}$ and A are not equivalent types. In other words, the following type is inhabited:*

$$\neg(\mathsf{Pow}_{i,j}\ \mathsf{A} \simeq \mathsf{A})$$

In particular, the LLInSAlg constructed by Corollary 3.2.22, is not a "fixpoint" for functor $\mathsf{Pow}_{i,j}$

Proof. Let $(\mathsf{A}, \mathsf{In}_\mathsf{A}, \mathsf{mc}) : \mathsf{LLInSAlg}_i\ \mathsf{Pow}_{i,j}$. Suppose, for a contradiction, that $\mathsf{Pow}_{i,j}\ \mathsf{A}$ and A are equivalent types.

Then, by definition of $\mathsf{Pow}_{i,j}$, types $\mathsf{A} \to \mathbf{Prop}_j$ and A are equivalent, which is impossible by Lemma 1.8.20. □

These results support the claim that LLInSAlgs are weaker than InSAlgs. Nevertheless, LLInSAlgs are still *useful*, as they can define *unique* functions by recursive equations (see Section 3.2.4).

This is a good place to emphasize a very important remark regarding the explicit use of universe levels. The definitions for LLInSAlg and InSAlg differ only by a universe level. If universe levels are omitted altogether on this chapter (as we did in Chapter 2), we are going to be puzzled by the apparent contradiction that every endofunctor has an initial set algebra, while at the same time we can construct an endofunctor without one! This justifies their explicit use.

We proceed to show an example on how lower level initial set H-algebras are used. In Section 3.2.5 we will discuss particularities and limitations of the approach leading to Corollary 3.2.22.

3.2.4 Example: Natural numbers

We will construct the natural numbers as a LLInSAlg. This will be a case where a LLInSAlg *is* an InSAlg.

First, we need to define the functor.

Definition 3.2.28. We define the function $\mathsf{NatF}_i : \mathcal{U}_i \to \mathcal{U}_i$ as:

$$\mathsf{NatF}_i :\equiv (\lambda \mathsf{Y} : \mathcal{U}_i.\ \mathbb{1} + \mathsf{Y}) : \mathcal{U}_i \to \mathcal{U}_i$$

Given $\mathsf{A}, \mathsf{B} : \mathcal{U}_i$ and function $\mathsf{f} : \mathsf{A} \to \mathsf{B}$, we define the mapping function as:

$$\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{f}\ (\mathsf{inl}\ \mathsf{l}) :\equiv \mathsf{inl}\ \mathsf{l}$$
$$\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{f}\ (\mathsf{inr}\ \mathsf{r}) :\equiv \mathsf{inr}\ (\mathsf{f}\ \mathsf{r})$$

having type:

$$\prod_{\mathsf{A},\mathsf{B}:\mathcal{U}_i} (\mathsf{A} \to \mathsf{B}) \to ((\mathbb{1} + \mathsf{A}) \to (\mathbb{1} + \mathsf{B}))$$

▲

Function $\mathsf{NatF}_i : \mathcal{U}_i \to \mathcal{U}_i$ is well-typed, as we explain next. Given $\mathsf{Y} : \mathcal{U}_i$, type $\mathbb{1} + \mathsf{Y}$ is at level i, because type $\mathbb{1}$ is at every universe level (see formation rule on page 42), and the sum type is at level $\max\{i, i\} = i$ (see Remark 1.5.37).

Function NatF_i and its mapping function form a functor.

Lemma 3.2.29. *The function* NatF_i *(together with its mapping function* $\mathsf{map}^{\mathsf{NatF}_i}$*) is an endofunctor that preserves sets.*

Proof. It is obvious from its definition that NatF_i is an endofunctor. NatF_i preserves sets because $\mathbb{1}$ is a set (Lemma 1.8.18), and sets are closed under sum types (see Lemma 1.8.18).

It remains to prove the two functorial properties (see Definition 1.9.6).

- Type $\prod_{\mathsf{A}:\mathcal{U}_i} \prod_{\mathsf{w}:\mathbb{1}+\mathsf{A}} \mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{w} = \mathsf{w}$ is inhabited.

  Let $\mathsf{A}:\mathcal{U}_i$. We will use the induction principle for sum types (Lemma 1.5.35) to prove that the following type is inhabited:

$$\prod_{\mathsf{w}:\mathbb{1}+\mathsf{A}} \mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{id}_{\mathsf{A}}\ \mathsf{w} = \mathsf{w}$$

  - "Left" Case.
    We are given $\mathsf{l}:\mathbb{1}$, and we need to prove $\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{id}_{\mathsf{A}}\ (\mathsf{inl}\ \mathsf{l}) = \mathsf{inl}\ \mathsf{l}$.
    But $\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{id}_{\mathsf{A}}\ (\mathsf{inl}\ \mathsf{l}) \equiv \mathsf{inl}\ \mathsf{l}$, by definition of $\mathsf{map}^{\mathsf{NatF}_i}$.
  - "Right" Case.
    We are given $\mathsf{r}:\mathsf{A}$, and we need to prove $\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{id}_{\mathsf{A}}\ (\mathsf{inr}\ \mathsf{r}) = \mathsf{inr}\ \mathsf{r}$.
    By definition, we have:

$$\begin{aligned}\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{id}_{\mathsf{A}}\ (\mathsf{inr}\ \mathsf{r}) &\equiv \mathsf{inr}\ (\mathsf{id}_{\mathsf{A}}\ \mathsf{r})\\ &\equiv \mathsf{inr}\ \mathsf{r}\end{aligned}$$

- The following type is inhabited:

$$\prod_{\mathsf{A},\mathsf{B},\mathsf{C}:\mathcal{U}_i}\ \prod_{\mathsf{g}:\mathsf{A}\to\mathsf{B}}\ \prod_{\mathsf{h}:\mathsf{B}\to\mathsf{C}}\ \prod_{\mathsf{w}:\mathbb{1}+\mathsf{A}} \mathsf{map}^{\mathsf{NatF}_i}\ (\mathsf{h}\circ\mathsf{g})\ \mathsf{w} = \mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ \mathsf{w})$$

  Let $\mathsf{A},\mathsf{B},\mathsf{C}:\mathcal{U}_i$, $\mathsf{g}:\mathsf{A}\to\mathsf{B}$, and $\mathsf{h}:\mathsf{B}\to\mathsf{C}$. Again, we will use the induction principle for sum types to prove that the following type is inhabited:

$$\prod_{\mathsf{w}:\mathbb{1}+\mathsf{A}} \mathsf{map}^{\mathsf{NatF}_i}\ (\mathsf{h}\circ\mathsf{g})\ \mathsf{w} = \mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ \mathsf{w})$$

  - "Left" Case.
    We are given $\mathsf{l}:\mathbb{1}$, and we need to prove:

$$\mathsf{map}^{\mathsf{NatF}_i}\ (\mathsf{h}\circ\mathsf{g})\ (\mathsf{inl}\ \mathsf{l}) = \mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ (\mathsf{inl}\ \mathsf{l}))$$

    But, by definition, we have:

$$\begin{aligned}\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ (\mathsf{inl}\ \mathsf{l})) &\equiv \mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}\ (\mathsf{inl}\ \mathsf{l})\\ &\equiv \mathsf{inl}\ \mathsf{l}\end{aligned}$$

    And also:

$$\mathsf{map}^{\mathsf{NatF}_i}\ (\mathsf{h}\circ\mathsf{g})\ (\mathsf{inl}\ \mathsf{l}) \equiv \mathsf{inl}\ \mathsf{l}$$

  - "Right" Case.
    We are given $\mathsf{r}:\mathsf{A}$, and we need to prove:

$$\mathsf{map}^{\mathsf{NatF}_i}\ (\mathsf{h}\circ\mathsf{g})\ (\mathsf{inr}\ \mathsf{r}) = \mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ (\mathsf{inr}\ \mathsf{r}))$$

    But, by definition, we have:

$$\begin{aligned}\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ (\mathsf{inr}\ \mathsf{r})) &\equiv \mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}\ (\mathsf{inr}\ (\mathsf{g}\ \mathsf{r}))\\ &\equiv \mathsf{inr}\ (\mathsf{h}\ (\mathsf{g}\ \mathsf{r}))\end{aligned}$$

    And also:

$$\mathsf{map}^{\mathsf{NatF}_i}\ (\mathsf{h}\circ\mathsf{g})\ (\mathsf{inr}\ \mathsf{r}) \equiv \mathsf{inr}\ (\mathsf{h}\ (\mathsf{g}\ \mathsf{r}))$$

□

Therefore, for $i > 0$, functor NatF_i has a LLInSAlg by Corollary 3.2.22. We introduce notation for this.

Definition 3.2.30. Let $i > 0$ be a universe level. We will denote by $(\mathsf{Nat}_i, \mathsf{In}_{\mathsf{Nat}_i}, \mathsf{mc})$ the lower level initial set NatF_i-algebra constructed by Corollary 3.2.22.

▲

Observe that the $\mathsf{In}_{\mathsf{Nat}_i}$ function has type $(\mathsf{NatF}_i\ \mathsf{Nat}_i) \to \mathsf{Nat}_i$, which is equivalent to $(\mathbb{1} + \mathsf{Nat}_i) \to \mathsf{Nat}_i$. Intuitively, this means we can "split" the $\mathsf{In}_{\mathsf{Nat}_i} : (\mathbb{1} + \mathsf{Nat}_i) \to \mathsf{Nat}_i$ function into two functions $\mathbb{1} \to \mathsf{Nat}_i$ and $\mathsf{Nat}_i \to \mathsf{Nat}_i$, which will be the so-called constructors for Nat_i. These constructors correspond to the zero constant and the successor function, respectively.

Definition 3.2.31. Let $i > 0$ be a universe level. We define two *constructors* for Nat_i.

- The *zero constant* $\overline{0}_i : \mathsf{Nat}_i$ is defined as:
$$\overline{0}_i :\equiv \mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inl}\ \star)$$
- The *successor function* $\overline{\mathsf{S}}_i : \mathsf{Nat}_i \to \mathsf{Nat}_i$ is defined as:
$$\overline{\mathsf{S}}_i :\equiv \lambda \mathsf{n} : \mathsf{Nat}_i.\ \mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inr}\ \mathsf{n})$$

▲

We proceed to prove the standard induction principle for the natural numbers.

Theorem 3.2.32. *(Induction Principle for Nat_i) Let $i > 0$ and $j \leq i$ be universe levels. Let $\mathsf{P} : \mathsf{Nat}_i \to \mathbf{Prop}_j$ be a predicate. If the following type is inhabited:*

$$(\mathsf{P}\ \overline{0}_i) \times \left(\prod_{\mathsf{n}:\mathsf{Nat}_i} (\mathsf{P}\ \mathsf{n}) \to (\mathsf{P}\ (\overline{\mathsf{S}}_i\ \mathsf{n})) \right)$$

then, the following type is inhabited:

$$\prod_{\mathsf{n}:\mathsf{Nat}_i} \mathsf{P}\ \mathsf{n}$$

Proof. Let:

$$\mathsf{base} : \mathsf{P}\ \overline{0}_i$$
$$\mathsf{Ind} : \prod_{\mathsf{n}:\mathsf{Nat}_i} (\mathsf{P}\ \mathsf{n}) \to (\mathsf{P}\ (\overline{\mathsf{S}}_i\ \mathsf{n}))$$

We will use Lemma 3.2.20 to conclude that the following type is inhabited:[27]

$$\prod_{\mathsf{n}:\mathsf{Nat}_i} \mathsf{P}\ \mathsf{n}$$

Hence, it is enough to prove that the following type is inhabited:

$$\prod_{\mathsf{T}:\mathsf{Nat}_i \to \mathbf{Prop}_j} \left(\prod_{\mathsf{w}:\sum_{\mathsf{y}:\mathsf{Nat}_i} \mathsf{T}\ \mathsf{y}} \mathsf{P}\ (\mathsf{pr}_1\ \mathsf{w}) \right) \to \left(\prod_{\mathsf{w}:\mathsf{NatF}_i \sum_{\mathsf{y}:\mathsf{Nat}_i} \mathsf{T}\ \mathsf{y}} \mathsf{P}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{Nat}_i}\ \mathsf{w}) \right)$$

Let $\mathsf{T} : \mathsf{Nat}_i \to \mathbf{Prop}_j$ with:

$$\mathsf{HInd} : \prod_{\mathsf{w}:\sum_{\mathsf{y}:\mathsf{Nat}_i} \mathsf{T}\ \mathsf{y}} \mathsf{P}\ (\mathsf{pr}_1\ \mathsf{w})$$

By definition of NatF_i, we need to prove:

$$\prod_{\mathsf{w}:\mathbb{1}+\sum_{\mathsf{y}:\mathsf{Nat}_i} \mathsf{T}\ \mathsf{y}} \mathsf{P}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{Nat}_i}\ \mathsf{w})$$

By the induction principle for sum types, it is enough to prove the following cases.

[27] Notice we can use Lemma 3.2.20 because Nat_i is of the form A_{I}, where A is the LLWinSAlg constructed by Lemma 3.2.7.

- "Left Case".
  We need to prove:
  $$\prod_{l:\mathbb{1}} \mathsf{P}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{Nat}_i}\ (\mathsf{inl}\ l))$$
  By the induction principle for the unit type $\mathbb{1}$ (Lemma 1.5.47), it is enough to prove:
  $$\mathsf{P}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{Nat}_i}\ (\mathsf{inl}\ \star))$$
  But, by definition of $\mathsf{In}^{\mathsf{T}}_{\mathsf{Nat}_i}$ (see Definition 3.2.12), the definition of $\mathsf{map}^{\mathsf{NatF}_i}$, and the definition of $\overline{0}_i$, we have:
  $$\begin{aligned}\mathsf{In}^{\mathsf{T}}_{\mathsf{Nat}_i}\ (\mathsf{inl}\ \star) &\equiv \mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{pr}_1\ (\mathsf{inl}\ \star))\\ &\equiv \mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inl}\ \star)\\ &\equiv \overline{0}_i\end{aligned}$$
  In other words, we need to prove that the following type is inhabited:
  $$\mathsf{P}\ \overline{0}_i$$
  but this is immediate by hypothesis $\mathsf{base}\colon \mathsf{P}\ \overline{0}_i$.
- "Right Case".
  We need to prove:
  $$\prod_{r:\sum_{y:\mathsf{Nat}_i} \mathsf{T}\ y} \mathsf{P}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{Nat}_i}\ (\mathsf{inr}\ r))$$
  Let $r\colon \sum_{y:\mathsf{Nat}_i} \mathsf{T}\ y$. Then, we have:
  $$\mathsf{HInd}\ r\colon \mathsf{P}\ (\mathsf{pr}_1\ r)$$
  Define:
  $$t_1 :\equiv \mathsf{Ind}\ (\mathsf{pr}_1\ r)\ (\mathsf{HInd}\ r)\colon \mathsf{P}\ (\overline{\mathsf{S}}_i\ (\mathsf{pr}_1\ r))$$
  Then, by definition of $\mathsf{In}^{\mathsf{T}}_{\mathsf{Nat}_i}$, the definition of $\mathsf{map}^{\mathsf{NatF}_i}$, and the definition of $\overline{\mathsf{S}}_i$, we have:
  $$\begin{aligned}\mathsf{In}^{\mathsf{T}}_{\mathsf{Nat}_i}\ (\mathsf{inr}\ r) &\equiv \mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{pr}_1\ (\mathsf{inr}\ r))\\ &\equiv \mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inr}\ (\mathsf{pr}_1\ r))\\ &\equiv \overline{\mathsf{S}}_i\ (\mathsf{pr}_1\ r)\end{aligned}$$
  In other words, we need to prove that the following type is inhabited:
  $$\mathsf{P}\ (\mathsf{In}^{\mathsf{T}}_{\mathsf{Nat}_i}\ (\mathsf{inr}\ r))$$
  which is equivalent to:
  $$\mathsf{P}\ (\overline{\mathsf{S}}_i\ (\mathsf{pr}_1\ r))$$
  but this is immediate from term $t_1\colon \mathsf{P}\ (\overline{\mathsf{S}}_i\ (\mathsf{pr}_1\ r))$.

□

Notice that this induction principle works for predicates (i.e. $\mathsf{P}\colon \mathsf{Nat}_i \to \mathbf{Prop}_j$), but not for general type families (i.e. $\mathsf{P}\colon \mathsf{Nat}_i \to \mathcal{U}_j$). This is one limitation of the approach in Section 3.2.2. Nevertheless, this limitation is not critical, as long as we work in the universe of propositions and sets.

Next, we prove the standard iteration principle for the natural numbers.

Theorem 3.2.33 (Iteration principle for Nat_i)**.** *Let $i > 0$ be a universe level. Let $\mathsf{D} : \mathbf{Set}_{i-1}$ be a set, $\mathsf{base} : \mathsf{D}$ a constant, and $\mathsf{step} : \mathsf{D} \to \mathsf{D}$ a function. Then, the following type is inhabited:*

$$\mathsf{IsContr}\left[\sum_{\mathsf{f}:\mathsf{Nat}_i \to \mathsf{D}} (\mathsf{f}\ \overline{0}_i = \mathsf{base}) \times \left(\prod_{\mathsf{n}:\mathsf{Nat}_i} \mathsf{f}\ (\overline{\mathsf{S}}_i\ \mathsf{n}) = \mathsf{step}\ (\mathsf{f}\ \mathsf{n})\right)\right] \tag{3.19}$$

In other words, there is a unique *function $\mathsf{f} : \mathsf{Nat}_i \to \mathsf{D}$ satisfying the stated recursive equations.*

Proof. First, by the recursion principle for sum types, we can define a function $\mathsf{In}_\mathsf{D} : (\mathsf{NatF}_i\ \mathsf{D}) \to \mathsf{D}$, or equivalently, $\mathsf{In}_\mathsf{D} : (\mathbb{1} + \mathsf{D}) \to \mathsf{D}$, as follows:

$$\mathsf{In}_\mathsf{D}\ (\mathsf{inl}\ \mathsf{l}) :\equiv \mathsf{base}$$
$$\mathsf{In}_\mathsf{D}\ (\mathsf{inr}\ \mathsf{r}) :\equiv \mathsf{step}\ \mathsf{r}$$

Therefore, $(\mathsf{D}, \mathsf{In}_\mathsf{D})$ is a set NatF_i-algebra at level $i-1$. Since Nat_i is a LLInSAlg, there is a *unique* morphism $\mathsf{h} : \mathsf{Nat}_i \to \mathsf{D}$ making the following diagram commute:[28]

$$\begin{array}{ccc} \mathbb{1} + \mathsf{Nat}_i & \xrightarrow{\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}} & \mathbb{1} + \mathsf{D} \\ {\scriptstyle \mathsf{In}_{\mathsf{Nat}_i}}\downarrow & & \downarrow{\scriptstyle \mathsf{In}_\mathsf{D}} \\ \mathsf{Nat}_i & \xrightarrow[\mathsf{h}]{} & \mathsf{D} \end{array} \tag{3.20}$$

To prove that type (3.19) is inhabited (see Definition 1.8.1), we have to construct a function f satisfying the stated equations (i.e. the existence step) such that any other function satisfying the equations must be equal to f (i.e. the uniqueness step).

Existence

We propose the morphism h above as our candidate function. It remains to construct a proof w_1 that h satisfies the equations in type (3.19).

But we have:

$$\begin{aligned} \mathsf{h}\ \overline{0}_i &\overset{(1)}{=} \mathsf{h}\ (\mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inl}\ \star)) \\ &\overset{(2)}{=} \mathsf{In}_\mathsf{D}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}\ (\mathsf{inl}\ \star)) \\ &\overset{(3)}{=} \mathsf{In}_\mathsf{D}\ (\mathsf{inl}\ \star) \\ &\overset{(4)}{=} \mathsf{base} \end{aligned}$$

where (1) follows by definition of $\overline{0}_i$, (2) by commutativity of Diagram (3.20), (3) by definition of $\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}$, and (4) by definition of In_D.

Also, given an arbitrary $\mathsf{n} : \mathsf{Nat}_i$, we have:

$$\begin{aligned} \mathsf{h}\ (\overline{\mathsf{S}}_i\ \mathsf{n}) &\overset{(1)}{=} \mathsf{h}\ (\mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inr}\ \mathsf{n})) \\ &\overset{(2)}{=} \mathsf{In}_\mathsf{D}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}\ (\mathsf{inr}\ \mathsf{n})) \\ &\overset{(3)}{=} \mathsf{In}_\mathsf{D}\ (\mathsf{inr}\ (\mathsf{h}\ \mathsf{n})) \\ &\overset{(4)}{=} \mathsf{step}\ (\mathsf{h}\ \mathsf{n}) \end{aligned}$$

[28] In more detail, given $(\mathsf{Nat}_i, \mathsf{In}_{\mathsf{Nat}_i}, \mathsf{mc})$, we have $\mathsf{mc}\ (\mathsf{D}, \mathsf{In}_\mathsf{D}) : \mathsf{IsContr}\ (\mathsf{AlgMor}\ \mathsf{Nat}_i\ \mathsf{D})$. In other words, the following type is inhabited:

$$\sum_{(\mathsf{h},\mathsf{q}_1):\mathsf{AlgMor}\ \mathsf{Nat}_i\ \mathsf{D}} \prod_{(\mathsf{g},\mathsf{q}_2):\mathsf{AlgMor}\ \mathsf{Nat}_i\ \mathsf{D}} (\mathsf{h}, \mathsf{q}_1) = (\mathsf{g}, \mathsf{q}_2)$$

This means we have a morphism $(\mathsf{h}, \mathsf{q}_1)$, where q_1 is a proof that the corresponding diagram for h commutes, and any other morphism $(\mathsf{g}, \mathsf{q}_2)$ must be equal to $(\mathsf{h}, \mathsf{q}_1)$.

where (1) follows by definition of $\overline{\mathsf{S}}_i$, (2) by commutativity of Diagram (3.20), (3) by definition of $\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{h}$, and (4) by definition of In_{D}.

Uniqueness

By definition of contractible type, we need to prove that the following type is inhabited:

$$\prod_{(\mathsf{g},\mathsf{w}_2):\sum_{\mathsf{f}:\mathsf{Nat}_i\to\mathsf{D}}(\mathsf{f}\ \overline{0}_i=\mathsf{base})\times\left(\prod_{\mathsf{n}:\mathsf{Nat}_i}\mathsf{f}\ (\overline{\mathsf{S}}_i\ \mathsf{n})=\mathsf{step}\ (\mathsf{f}\ \mathsf{n})\right)} (\mathsf{h},\mathsf{w}_1)=(\mathsf{g},\mathsf{w}_2)$$

where w_1 is a proof (constructed in the existence part) that h satisfies the equations, and w_2 is a proof that g satisfies the equations.

So, let $\mathsf{g}\colon \mathsf{Nat}_i \to \mathsf{D}$ satisfying the recursive equations:

$$(\mathsf{g}\ \overline{0}_i = \mathsf{base}) \times \left(\prod_{\mathsf{n}:\mathsf{Nat}_i} \mathsf{g}\ (\overline{\mathsf{S}}_i\ \mathsf{n}) = \mathsf{step}\ (\mathsf{g}\ \mathsf{n})\right)$$

We know that the second component of the sigma type:

$$\sum_{\mathsf{f}:\mathsf{Nat}_i\to\mathsf{D}} (\mathsf{f}\ \overline{0}_i = \mathsf{base}) \times \left(\prod_{\mathsf{n}:\mathsf{Nat}_i} \mathsf{f}\ (\overline{\mathsf{S}}_i\ \mathsf{n}) = \mathsf{step}\ (\mathsf{f}\ \mathsf{n})\right)$$

(i.e. the recursive equations) is a mere proposition for every $\mathsf{f}\colon \mathsf{Nat}_i \to \mathsf{D}$, since all terms appearing in the equalities are of type D (which is a set), and propositions are closed under the product type and Π-types (Lemma 1.8.17).

Therefore, by Lemma 1.8.14, to prove that $(\mathsf{h},\mathsf{w}_1) = (\mathsf{g},\mathsf{w}_2)$ is inhabited, it is enough to prove that $\mathsf{h} = \mathsf{g}$ is inhabited.

To prove $\mathsf{h} = \mathsf{g}$, we are going to show that g is an algebra morphism from Nat_i to D, because h is the *only* such morphism by definition of LLInSAlg.

So, we need to construct a proof q_2 that the following diagram commutes:

$$\begin{array}{ccc} \mathbb{1} + \mathsf{Nat}_i & \xrightarrow{\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}} & \mathbb{1} + \mathsf{D} \\ {\scriptstyle \mathsf{In}_{\mathsf{Nat}_i}}\downarrow & & \downarrow{\scriptstyle \mathsf{In}_{\mathsf{D}}} \\ \mathsf{Nat}_i & \xrightarrow[\mathsf{g}]{} & \mathsf{D} \end{array}$$

By the induction principle for sum types, it is enough to prove the following cases.

- "Left Case".

  We need to prove:

  $$\prod_{\mathsf{l}:\mathbb{1}} \mathsf{g}\ (\mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inl}\ \mathsf{l})) = \mathsf{In}_{\mathsf{D}}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ (\mathsf{inl}\ \mathsf{l}))$$

  By the induction principle for the unit type $\mathbb{1}$, it is enough to prove:

  $$\mathsf{g}\ (\mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inl}\ \star)) = \mathsf{In}_{\mathsf{D}}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ (\mathsf{inl}\ \star))$$

  But we have:

  $$\begin{aligned} \mathsf{g}\ (\mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inl}\ \star)) &\overset{(1)}{=} \mathsf{g}\ \overline{0}_i \\ &\overset{(2)}{=} \mathsf{base} \end{aligned}$$

  where (1) follows by definition of $\overline{0}_i$, and (2) by hypothesis on function g.

And also:

$$\begin{aligned}\mathsf{In}_{\mathsf{D}}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ (\mathsf{inl}\ \star)) &\overset{(1)}{=} \mathsf{In}_{\mathsf{D}}\ (\mathsf{inl}\ \star)\\ &\overset{(2)}{=} \mathsf{base}\end{aligned}$$

where (1) follows by definition of $\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}$, and (2) by definition of In_{D}.

- "Right Case".

  We are given $\mathsf{r} : \mathsf{Nat}_i$, and we need to prove:

$$\mathsf{g}\ (\mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inr}\ \mathsf{r})) = \mathsf{In}_{\mathsf{D}}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ (\mathsf{inr}\ \mathsf{r}))$$

  But we have:

$$\begin{aligned}\mathsf{g}\ (\mathsf{In}_{\mathsf{Nat}_i}\ (\mathsf{inr}\ \mathsf{r})) &\overset{(1)}{=} \mathsf{g}\ (\overline{\mathsf{S}}_i\ \mathsf{r})\\ &\overset{(2)}{=} \mathsf{step}\ (\mathsf{g}\ \mathsf{r})\end{aligned}$$

  where (1) follows by definition of $\overline{\mathsf{S}}_i$, and (2) by hypothesis on function g.

  And also:

$$\begin{aligned}\mathsf{In}_{\mathsf{D}}\ (\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}\ (\mathsf{inr}\ \mathsf{r})) &\overset{(1)}{=} \mathsf{In}_{\mathsf{D}}\ (\mathsf{inr}\ (\mathsf{g}\ \mathsf{r}))\\ &\overset{(2)}{=} \mathsf{step}\ (\mathsf{g}\ \mathsf{r})\end{aligned}$$

  where (1) follows by definition of $\mathsf{map}^{\mathsf{NatF}_i}\ \mathsf{g}$, and (2) by definition of In_{D}.

□

Notice that Theorem 3.2.33 requires D to be a set at level $i-1$. This level restriction is forced by the fact that Nat_i is a LLInSAlg. If we attempt to prove Theorem 3.2.33, but this time for a set D at level i, we will not be able to construct the morphism $\mathsf{h} : \mathsf{Nat}_i \to \mathsf{D}$. The reason is that Nat_i expects an algebra at level $i-1$, but $(\mathsf{D}, \mathsf{In}_{\mathsf{D}})$ will be at level i. This small detail has very interesting consequences, specially when defining functions on Nat_i.

For example, let us try to define addition on the natural numbers $\mathsf{add}_i : \mathsf{Nat}_i \to \mathsf{Nat}_i \to \mathsf{Nat}_i$ for some $i > 0$. We want to use the iteration principle, but the universe level restriction forces us to define addition as having type $\mathsf{add}_i : \mathsf{Nat}_i \to \mathsf{Nat}_{i+1} \to \mathsf{Nat}_i$. Let us do it step by step.

We are going to do recursion on the second parameter of add_i. Define:

$$\mathsf{add}_i :\equiv \lambda \mathsf{n} : \mathsf{Nat}_i.\ \alpha_{\mathsf{n}}$$

where $\alpha_{\mathsf{n}} : \mathsf{Nat}_{i+1} \to \mathsf{Nat}_i$ is a function (depending on n) we still need to construct. We will use the iteration principle to build function α_{n}.

First, we require a constant of type Nat_i. We have $\mathsf{n} : \mathsf{Nat}_i$ available.

Second, we require a function of type $\mathsf{Nat}_i \to \mathsf{Nat}_i$. We have the successor function $\overline{\mathsf{S}}_i$.

Therefore, by the iteration principle, there is a unique function α_{n} satisfying:

$$\alpha_{\mathsf{n}}\ \overline{0}_{i+1} = \mathsf{n}$$

and:

$$\prod_{\mathsf{m}:\mathsf{Nat}_{i+1}} \alpha_{\mathsf{n}}\ (\overline{\mathsf{S}}_{i+1}\ \mathsf{m}) = \overline{\mathsf{S}}_i\ (\alpha_{\mathsf{n}}\ \mathsf{m})$$

If we write $\mathsf{add}_i\ \mathsf{n}$ instead of α_{n}, all these equations can be written as:

$$\prod_{\mathsf{n}:\mathsf{Nat}_i} \mathsf{add}_i\ \mathsf{n}\ \overline{0}_{i+1} = \mathsf{n}$$

$$\prod_{\mathsf{n}:\mathsf{Nat}_i} \prod_{\mathsf{m}:\mathsf{Nat}_{i+1}} \mathsf{add}_i\ \mathsf{n}\ (\overline{\mathsf{S}}_{i+1}\ \mathsf{m}) = \overline{\mathsf{S}}_i\ (\mathsf{add}_i\ \mathsf{n}\ \mathsf{m})$$

This example shows that we need to play a lot with universe levels. However, we quickly find difficulties. For example, there is a *down cast* function $\downarrow\colon \mathsf{Nat}_{i+1} \to \mathsf{Nat}_i$, defined by the equations:

$$\begin{aligned} \downarrow \overline{0}_{i+1} &= \overline{0}_i \\ \prod_{n:\mathsf{Nat}_{i+1}} \downarrow (\overline{S}_{i+1}\ n) &= \overline{S}_i\ (\downarrow n) \end{aligned}$$

The down cast function behaves like an "identity function" mapping each number to its corresponding number at a lower level. Is there an *up cast* function $\uparrow\colon \mathsf{Nat}_i \to \mathsf{Nat}_{i+1}$? Clearly, we cannot use the iteration principle, because the codomain type is at a higher level than the domain type. Nevertheless, both types Nat_i and Nat_{i+1} *look the same*: both have a zero and every other number can be constructed by using the successor function. Intuitively, these two types *should* be equivalent. In fact, if we can build the up cast function, it follows that all Nat_i at every level become mutually equivalent. If all types Nat_i for every $i > 0$ are equivalent, we do not have to worry about universe levels when defining functions between them.

One possible approach for defining the up cast function is to use the principle of unique choice (Lemma 1.11.8).[29] Another approach is to construct the function as a fixpoint of partial functions, by using techniques similar to those on Chapter 2.

Instead of building the up cast function, we will prove that all types Nat_i (for $i > 0$) are equivalent, by proving that each one of them is equivalent to the natural numbers type $\mathbb{N}$ (see Section 1.5.6). This will show that Nat_i is actually an initial set NatF_i-algebra, and one consequence of this will be the existence of the up cast function.

Also, proving that Nat_i is equivalent to $\mathbb{N}$ highlights an important observation. If we add to HoTT inference rules for working with the inductive type generated by some functor H, we will find out that the lower level initial algebra for functor H will be equivalent to the inductive type introduced by the inference rules. It is still unknown to the author to what extend this observation is true. However, Theorem 3.2.25 should warn the reader that we cannot indiscriminately add inference rules into HoTT. For example, if we add inference rules for working with the "inductive type" of the powertype functor, we will be implicitly adding an initial algebra for the powertype functor, contradicting Theorem 3.2.25 and producing, as a result, an inconsistent type theory.

Theorem 3.2.34. *Let $i > 0$ be a universe level. Then, types Nat_i and $\mathbb{N}$ are equivalent. In other words, the following type is inhabited:*

$$\mathsf{Nat}_i \simeq \mathbb{N}$$

Proof. We have to define two functions $\mathsf{f}\colon \mathsf{Nat}_i \to \mathbb{N}$ and $\mathsf{g}\colon \mathbb{N} \to \mathsf{Nat}_i$, so that f and g are inverses of each other.

To define function $\mathsf{f}\colon \mathsf{Nat}_i \to \mathbb{N}$, we will use the iteration principle for Nat_i. We can use the iteration principle because $\mathbb{N}$ is at every universe level (see formation rule for $\mathbb{N}$ on page 42). In particular, $\mathbb{N}$ is at level $i - 1$. Also, by Lemma 1.8.18, $\mathbb{N}$ is a set.

So, there is a unique function $\mathsf{f}\colon \mathsf{Nat}_i \to \mathbb{N}$, defined by the equations:

$$\begin{aligned} \mathsf{f}\ \overline{0}_i &= 0 \\ \prod_{n:\mathsf{Nat}_i} \mathsf{f}\ (\overline{S}_i\ n) &= \mathsf{succ}\ (\mathsf{f}\ n) \end{aligned}$$

For the other function $\mathsf{g}\colon \mathbb{N} \to \mathsf{Nat}_i$, we use the recursion principle for $\mathbb{N}$ as follows (for any $\mathsf{n}\colon \mathbb{N}$):

$$\begin{aligned} \mathsf{g}\ 0 &:\equiv \overline{0}_i \\ \mathsf{g}\ (\mathsf{succ}\ n) &:\equiv \overline{S}_i\ (\mathsf{g}\ n) \end{aligned}$$

To prove:

$$\prod_{n:\mathsf{Nat}_i} \mathsf{g}\ (\mathsf{f}\ n) = n$$

we use the induction principle for Nat_i. Notice we can use the induction principle, because $\mathsf{g}\ (\mathsf{f}\ n) = n$ is a proposition at level i for every $n\colon \mathsf{Nat}_i$, as Nat_i is a set at level i.

[29] However, the author has not explored this possibility.

For the base case, we have by definition:

$$\begin{aligned} \mathsf{g}\ (\mathsf{f}\ \overline{0}_i) &= \mathsf{g}\ 0 \\ &= \overline{0}_i \end{aligned}$$

For the inductive case, we suppose $\mathsf{g}\ (\mathsf{f}\ \mathsf{n}) = \mathsf{n}$ for some $\mathsf{n} : \mathsf{Nat}_i$, and we have:

$$\begin{aligned} \mathsf{g}\ (\mathsf{f}\ (\overline{\mathsf{S}}_i\ \mathsf{n})) &= \mathsf{g}\ (\mathsf{succ}\ (\mathsf{f}\ \mathsf{n})) \\ &= \overline{\mathsf{S}}_i\ (\mathsf{g}\ (\mathsf{f}\ \mathsf{n})) \\ &\overset{(1)}{=} \overline{\mathsf{S}}_i\ \mathsf{n} \end{aligned}$$

where (1) follows by the inductive hypothesis, and the rest follows by definition of f and g.

To prove:

$$\prod_{\mathsf{n}:\mathbb{N}} \mathsf{f}\ (\mathsf{g}\ \mathsf{n}) = \mathsf{n}$$

we use the induction principle for $\mathbb{N}$.

For the base case, we have by definition:

$$\begin{aligned} \mathsf{f}\ (\mathsf{g}\ 0) &= \mathsf{f}\ \overline{0}_i \\ &= 0 \end{aligned}$$

For the inductive case, we suppose $\mathsf{f}\ (\mathsf{g}\ \mathsf{n}) = \mathsf{n}$ for some $\mathsf{n} : \mathbb{N}$, and we have:

$$\begin{aligned} \mathsf{f}\ (\mathsf{g}\ (\mathsf{succ}\ \mathsf{n})) &= \mathsf{f}\ (\overline{\mathsf{S}}_i\ (\mathsf{g}\ \mathsf{n})) \\ &= \mathsf{succ}\ (\mathsf{f}\ (\mathsf{g}\ \mathsf{n})) \\ &\overset{(1)}{=} \mathsf{succ}\ \mathsf{n} \end{aligned}$$

where (1) follows by the inductive hypothesis, and the rest follows by definition of f and g.

□

Theorem 3.2.34 implies that all types Nat_i for $i > 0$ are mutually equivalent. Also, by Theorem 3.2.34 and univalence, we have $\mathsf{Nat}_i = \mathbb{N}$ for every $i > 0$.

It is not hard to prove that $\mathbb{N}$ can be given the structure of an initial set NatF_i-algebra. Define the $\mathsf{In}_{\mathbb{N}} : \mathsf{NatF}_i\ \mathbb{N} \to \mathbb{N}$ function by cases:

$$\begin{aligned} \mathsf{In}_{\mathbb{N}}\ (\mathsf{inl}\ \mathsf{l}) &:\equiv 0 \\ \mathsf{In}_{\mathbb{N}}\ (\mathsf{inr}\ \mathsf{r}) &:\equiv \mathsf{succ}\ \mathsf{r} \end{aligned}$$

and then, by the recursion principle for $\mathbb{N}$, we can construct a morphism from $\mathbb{N}$ to any other set NatF_i-algebra. Finally, by using the induction principle for $\mathbb{N}$, we can prove that such morphism is unique (we will not do the details, as it is not critical for the example).

Once we know that $\mathbb{N}$ is an InSAlg, we can transfer the InSAlg structure of $\mathbb{N}$ into algebra Nat_i, since we already know a proof $\mathsf{q} : \mathsf{Nat}_i = \mathbb{N}$ (by univalence). To transfer the structure, we use the $\mathsf{transport}$ function as follows.

First, we transfer the $\mathsf{In}_{\mathbb{N}}$ function:

$$\mathsf{In}'_{\mathsf{Nat}_i} :\equiv (\mathsf{transport}^{(\lambda \mathsf{Y}.\ \mathsf{NatF}_i\ \mathsf{Y} \to \mathsf{Y})}\ \mathsf{q}^{-1}\ \mathsf{In}_{\mathbb{N}}) : \mathsf{NatF}_i\ \mathsf{Nat}_i \to \mathsf{Nat}_i$$

Hence, by Lemma 1.6.25, we have a proof:

$$\mathsf{q}' : (\mathbb{N}, \mathsf{In}_{\mathbb{N}}) =_{(\mathsf{SAlg}_i\ \mathsf{NatF}_i)} (\mathsf{Nat}_i, \mathsf{In}'_{\mathsf{Nat}_i})$$

that $\mathbb{N}$ and Nat_i are identifiable *as algebras*.

Let us define a predicate on NatF_i-algebras:

$$\mathsf{P} :\equiv \lambda(\mathsf{A}, \mathsf{In}_\mathsf{A}) : \mathsf{SAlg}_i\ \mathsf{NatF}_i.\ \prod_{(\mathsf{B},\mathsf{In}_\mathsf{B}):\mathsf{SAlg}_i\ \mathsf{NatF}_i} \mathsf{IsContr}\ (\mathsf{AlgMor}\ \mathsf{A}\ \mathsf{B})$$

Since we know $\mathbb{N}$ is an InSAlg, we have a proof $\mathsf{mc} : \mathsf{P}\ (\mathbb{N}, \mathsf{In}_\mathbb{N})$. This means we can transfer mc to Nat_i by using the $\mathsf{transport}$ function and proof q' as follows:

$$\mathsf{mc}' :\equiv (\mathsf{transport}^\mathsf{P}\ \mathsf{q}'\ \mathsf{mc}) : \mathsf{P}\ (\mathsf{Nat}_i, \mathsf{In}'_{\mathsf{Nat}_i})$$

or equivalently:

$$\mathsf{mc}' :\equiv (\mathsf{transport}^\mathsf{P}\ \mathsf{q}'\ \mathsf{mc}) : \prod_{(\mathsf{B},\mathsf{In}_\mathsf{B}):\mathsf{SAlg}_i\ \mathsf{NatF}_i} \mathsf{IsContr}\ (\mathsf{AlgMor}\ \mathsf{Nat}_i\ \mathsf{B})$$

Hence, we have:

$$(\mathsf{Nat}_i, \mathsf{In}'_{\mathsf{Nat}_i}, \mathsf{mc}') : \mathsf{InSAlg}_i\ \mathsf{NatF}_i$$

In other words, Nat_i *can* be given the structure of an initial set NatF_i-algebra.

Recapitulating, we showed that all types Nat_i are mutually equivalent by proving that each Nat_i is equivalent to the natural numbers type $\mathbb{N}$. As a byproduct, each type Nat_i can be given the structure of an InSAlg. But how can we prove mutual equivalence between LLInSAlgs when there is no predefined type (like the natural numbers type $\mathbb{N}$ in our example)?

A first attempt involves the addition of inference rules to induce an inductive type, to which the LLInSAlgs will be equivalent. However, this approach is risky, as Theorem 3.2.25 suggests. There are functors without an initial algebra. Hence, we cannot add inference rules to induce an initial algebra for those functors, as we would be producing an inconsistent type theory.

A second approach involves the construction of an *up cast* function $\uparrow$ (see discussion before Theorem 3.2.34).

Even more interesting, it *seems* that the existence of an up cast function is another way to characterize initial algebras, i.e. the down cast function $\downarrow$ is an equivalence (having as inverse the up cast function $\uparrow$) *if and only if* Nat_i can be given the structure of an initial set NatF_i-algebra. This is mere speculation of course, but it justifies further investigation on the existence of an up cast function and its consequences.

3.2.5 Discussion

The first obvious limitation of the approach developed on Sections 3.2.1 and 3.2.2 is that it only builds *set* algebras. In what follows, we explain why we had to limit the development to sets.

The heart of the problem resides in our dependence on the results of Chapter 2. At step 1 of Section 3.2.2, we defined type A_I (Definition 3.2.17) as a *least fixpoint* on the powertype lattice of Example 2.2.8.

This causes the induction principle for A_I (Lemma 3.2.20) to work for predicates $\mathsf{A}_\mathsf{I} \to \mathbf{Prop}_m$, but *not* for general type families $\mathsf{A}_\mathsf{I} \to \mathcal{U}_m$. The reason is that the proof for the induction principle depends on Corollary 2.4.1, which is instantiated on the powertype lattice.

This restriction on the induction principle forces A_I to be a set, as we explain next. Let us suppose we want to use the induction principle to prove the statement $\prod_{\mathsf{w}:\mathsf{A}_\mathsf{I}} \mathsf{r}\ \mathsf{w} = \mathsf{t}\ \mathsf{w}$, where $\mathsf{r}, \mathsf{t} : \mathsf{A}_\mathsf{I} \to \mathsf{A}_\mathsf{I}$ are arbitrary functions. Since our induction principle only works for predicates, type $\mathsf{r}\ \mathsf{w} = \mathsf{t}\ \mathsf{w}$ is required to be a mere proposition for every $\mathsf{w} : \mathsf{A}_\mathsf{I}$. One way to ensure that $\mathsf{r}\ \mathsf{w} = \mathsf{t}\ \mathsf{w}$ is a mere proposition, is to *force* A_I to be a set.

For example, a situation like this is found in the proof of Theorem 3.2.34, where type $\prod_{\mathsf{n}:\mathsf{Nat}_i} \mathsf{g}\ (\mathsf{f}\ \mathsf{n}) = \mathsf{n}$ is proved by the induction principle for Nat_i (which is obtained directly from the induction principle for A_I).

Another restriction forced on us by the induction principle becomes evident in the proof of Theorem 3.2.21. There, the following type is proved by the induction principle for A_I:

$$\prod_{\mathsf{w}:\mathsf{A}_\mathsf{I}} \mathsf{h}\ \mathsf{w} = \mathsf{g}\ \mathsf{w}$$

where $\mathsf{h}, \mathsf{g} : \mathsf{A}_\mathsf{I} \to \mathsf{B}$, and B is the universally quantified set H-algebra in Definition 3.2.9. We observe that B *must* be a set, otherwise the identity $\mathsf{h}\ \mathsf{w} = \mathsf{g}\ \mathsf{w}$ is not guaranteed to be a mere proposition.

Therefore, we have two restrictions so far:

- $\mathsf{A_I}$ must be a set.
- H-algebra B in Definition 3.2.9 must be a set.

This justifies the restriction of Definitions 1.9.11 and 1.9.15 to *set* H-algebras and initial *set* H-algebras, as stated in Definitions 3.2.4 and 3.2.9.

Now, remember that $\mathsf{A_I}$ was defined in 3.2.17 as:

$$\sum_{\mathsf{w:A}} \mathsf{Lfp\ CanStep^A\ w}$$

On this definition, it is true that $\mathsf{A_I}$ *might* be a set even if A is not a set (it depends on how A interacts with the predicate $\mathsf{Lfp\ CanStep^A}$). To ensure that $\mathsf{A_I}$ is a set without doubt, we *force* A to be a set, ensuring this way that $\mathsf{A_I}$ is a set by Lemma 1.8.18.

Therefore, for Theorem 3.2.21 to work, the (lower level) weakly initial set H-algebra A in the hypothesis *must* be a set, because the theorem produces as initial algebra $\mathsf{A_I}$ and we already explained that $\mathsf{A_I}$ is forced to be a set (which then forces A to be a set).

Hence, we have the new restriction:

- Our (lower level) weakly initial H-algebras[30] must be sets.

This restriction justifies Definition 3.2.5 for (lower level) weakly initial *set* H-algebras.

The reader might think that Definition 3.2.5 does not need to be restricted to sets, because it is a reasonable guess to suppose that 0-truncating a (lower level) weakly initial H-algebra will produce a (lower level) weakly initial *set* H-algebra. However, it appears that functor H needs to satisfy the "morphism 0-truncation property" defined below for 0-truncation to work.

Although the "morphism 0-truncation property" is a sufficient condition, it is unknown to the author if it is a *necessary* condition, i.e. whether or not the following statement is true: "If 0-truncating a weakly initial H-algebra produces a weakly initial set H-algebra, then functor H must have the property".

Definition 3.2.35 (Morphism 0-truncation property)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor and $k, m \leq i$ universe levels.

We say that H 0-*truncates morphisms* if given an H-algebra $(\mathsf{A}, \mathsf{In_A}) : \mathsf{Alg}_k\ \mathsf{H}$, we can construct a function $\mathsf{In}_{\|\mathsf{A}\|_0} : (\mathsf{H}\ \|\mathsf{A}\|_0) \to \|\mathsf{A}\|_0$ such that for any other set H-algebra $(\mathsf{B}, \mathsf{In_B}) : \mathsf{SAlg}_m\ \mathsf{H}$ and algebra morphism $\mathsf{h} : \mathsf{A} \to \mathsf{B}$, we have that $\|\mathsf{h}\|_0 : \|\mathsf{A}\|_0 \to \mathsf{B}$ is an algebra morphism. Term $\|\mathsf{h}\|_0$ denotes the function produced by the recursion principle for 0-truncations when it is applied with function h (see Corollary 1.11.14).

▲

To see that the "morphism 0-truncation property" is a sufficient condition, let us suppose functor H has the property. Let us also suppose we are given a (lower level) weakly initial H-algebra $(\mathsf{W}, \mathsf{In_W}, \mathsf{m})$. Then, by the 0-truncation property, we have a *set* H-algebra $(\|\mathsf{W}\|_0, \mathsf{In}_{\|\mathsf{W}\|_0})$. Now, given an arbitrary set H-algebra $(\mathsf{B}, \mathsf{In_B})$, since W is (lower level) weakly initial, there is an algebra morphism $\mathsf{h} : \mathsf{W} \to \mathsf{B}$. Therefore, by the 0-truncation property, function $\|\mathsf{h}\|_0 : \|\mathsf{W}\|_0 \to \mathsf{B}$ is an algebra morphism. Hence, $\|\mathsf{W}\|_0$ is a (lower level) weakly initial *set* H-algebra.

However, in the proof for Lemma 3.2.7, we want to obtain a lower level weakly initial set H-algebra without imposing extra conditions on the functor. Therefore, we need to find a way to get lower level weakly initial set H-algebras without using 0-truncations.

It turns out that we can achieve this by taking advantage of the set closure property for Π-types in Lemma 1.8.18. In the proof for Lemma 3.2.7, we were able to construct a lower level weakly initial set H-algebra by taking the weak limit of the family of *set* H-algebras:

$$\mathsf{pr_1} : (\mathsf{SAlg}_{i-1}\ \mathsf{H}) \to \mathcal{U}_{i-1}$$

producing type:

$$\prod_{\mathsf{A:SAlg}_{i-1}\ \mathsf{H}} \mathsf{pr_1\ A}$$

[30] (Lower level) weakly initial H-algebras are like in Definition 3.2.5, but all references to SAlg are replaced with Alg (Definition 1.9.11).

which *is* a set by Lemma 1.8.18, because $\mathsf{pr}_1\ \mathsf{A}$ is a set by construction.

If instead we take the weak limit of the family of H-algebras (as one would do if the Knaster-Tarski construction is generalized directly):

$$\mathsf{pr}_1 : (\mathsf{Alg}_{i-1}\ \mathsf{H}) \to \mathcal{U}_{i-1}$$

we get type:

$$\prod_{\mathsf{A}:\mathsf{Alg}_{i-1}\ \mathsf{H}} \mathsf{pr}_1\ \mathsf{A}$$

which is *not* necessarily a set, as $\mathsf{pr}_1\ \mathsf{A}$ is a general type.

This small trick (i.e. taking advantage of Lemma 1.8.18) allowed the construction of lower level weakly initial set H-algebras without the need to use 0-truncations.

Therefore, if we hope to generalize the results of Section 3.2 to arbitrary (lower level) initial H-algebras and not just (lower level) initial *set* H-algebras, we have to find a way to remove the dependence on Chapter 2, as this is the main bottleneck that cascades to the results in Section 3.2.2.

The second obvious limitation is that our experiment only produces *lower level* initial set algebras, which are weaker than initial set algebras. Nevertheless, lower level initial set algebras are still useful concepts, as they are capable of defining *unique* functions by recursive equations.

3.3 Coinductive types

This section describes the construction of coinductive types by means of generalizing the Knaster-Tarski construction (Lemma 2.3.6(ii)) into the language of coalgebras.

As it happened with inductive types, we will see that generalizing Knaster-Tarski produces lower level *weakly* final coalgebras. Therefore, to obtain a (lower level) final coalgebra from a (lower level) weakly final coalgebra, we need to devise a method of "refinement".

Hence, the development will be split into two parts. Subsection 3.3.1 carries out the generalization of Knaster-Tarski, and Subsection 3.3.2 carries out the "refinement" of (lower level) weakly final coalgebras into (lower level) final coalgebras.

3.3.1 Generalizing the Knaster-Tarski construction

If we examine the proof for the second part of Knaster-Tarski (Lemma 2.3.6(ii)), we see that the greatest fixpoint was defined as the supremum of all postfixpoints of the monotone function. Hence, continuing the generalization started on Section 3.2.1, the only missing ingredient is a generalization to the concept of "least upper bound".

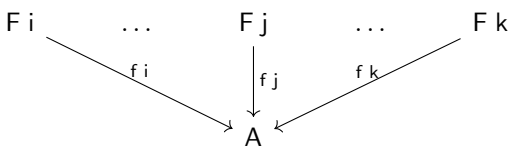
We define an "upper bound" for a family $\mathsf{F} : \mathsf{M} \to \mathcal{U}_j$ to be a type $\mathsf{A} : \mathcal{U}_k$ that is "above" every element in the family (i.e. for every $\mathsf{i} : \mathsf{M}$, we have a proof for $(\mathsf{F}\ \mathsf{i}) \to \mathsf{A}$, or interpreted order-theoretically, a proof for "$\mathsf{F}\ \mathsf{i} \leq \mathsf{A}$"). By borrowing terminology from category theory, this generalized notion of "upper bound" corresponds to the notion of *cocone* over the family F.

Definition 3.3.1 (Cocone over a family)**.** Let $\mathsf{M} : \mathcal{U}_i$ be an indexing type, and $\mathsf{F} : \mathsf{M} \to \mathcal{U}_j$ a family of types. Let k be a universe level. Any term of type:

$$\mathsf{Cocone}_k\ \mathsf{F} :\equiv \sum_{\mathsf{A}:\mathcal{U}_k} \prod_{\mathsf{i}:\mathsf{M}} (\mathsf{F}\ \mathsf{i}) \to \mathsf{A}$$

will be called a *cocone* over the family F at level k. ▲

Intuitively, $\mathsf{Cocone}_k\ \mathsf{F}$ is the type of all "upper bounds" for the family F. Observe that a cocone over F is a type A *together* with a family of functions $\mathsf{f} : \prod_{\mathsf{i}:\mathsf{M}} (\mathsf{F}\ \mathsf{i}) \to \mathsf{A}$. In other words, we have the diagram:

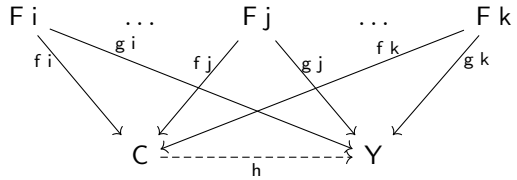


In Section 3.2.1, weak limits (Definition 3.2.2) served as a working generalization to the notion of "greatest lower bound". On this section, weak colimits will serve as a working generalization to the notion of "least upper bound".[31]

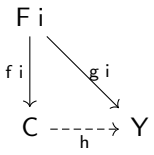
Definition 3.3.2 (Weak colimit of a type family)**.** Let $\mathsf{M} : \mathcal{U}_i$ be an indexing type, and $\mathsf{F} : \mathsf{M} \to \mathcal{U}_j$ a family of types. Let k be a universe level. Any term of type:

$$\mathsf{WCoLimit}_k\ \mathsf{F} :\equiv \sum_{(\mathsf{C},\mathsf{f}):\mathsf{Cocone}_k\ \mathsf{F}} \prod_{(\mathsf{Y},\mathsf{g}):\mathsf{Cocone}_k\ \mathsf{F}} \sum_{\mathsf{h}:\mathsf{C}\to\mathsf{Y}} \prod_{\mathsf{i}:\mathsf{M}} \prod_{\mathsf{w}:\mathsf{F}\ \mathsf{i}} \mathsf{g}\ \mathsf{i}\ \mathsf{w} = \mathsf{h}\ (\mathsf{f}\ \mathsf{i}\ \mathsf{w})$$

will be called a *weak colimit* for the family F. In other words, we have the commutative diagram:



i.e. the cocone (C, f) will be a weak colimit for F if for any other cocone (Y, g), there is a function $\mathsf{h} : \mathsf{C} \to \mathsf{Y}$ such that for every $\mathsf{i} : \mathsf{M}$, the following triangle commutes:



▲

In other words, C is a "least upper bound" for F if C is an "upper bound" for F (i.e. it forms a cocone for F), and any other "upper bound" Y (i.e. a cocone over F) is "above" C. The "commutativity condition" $\prod_{\mathsf{i}:\mathsf{M}} \prod_{\mathsf{w}:\mathsf{F}\ \mathsf{i}} \mathsf{g}\ \mathsf{i}\ \mathsf{w} = \mathsf{h}\ (\mathsf{f}\ \mathsf{i}\ \mathsf{w})$ expresses that functions $\mathsf{f} : \prod_{\mathsf{i}:\mathsf{M}} (\mathsf{F}\ \mathsf{i}) \to \mathsf{C}$, $\mathsf{g} : \prod_{\mathsf{i}:\mathsf{M}} (\mathsf{F}\ \mathsf{i}) \to \mathsf{Y}$, and $\mathsf{h} : \mathsf{C} \to \mathsf{Y}$ are not chosen arbitrarily.

The first result states that any family has a weak colimit (i.e. any family has a "least upper bound").

Lemma 3.3.3. *Let $\mathsf{M} : \mathcal{U}_i$ be an indexing type, and $\mathsf{F} : \mathsf{M} \to \mathcal{U}_j$ a family of types. Then, the following type is inhabited:*

$$\mathsf{WCoLimit}_{\max\{i,j\}}\ \mathsf{F}$$

Proof. First, we need to build a tuple $(\mathsf{C}, \mathsf{f}) : \mathsf{Cocone}_{\max\{i,j\}}\ \mathsf{F}$.

In standard mathematics, the least upper bound for a family of sets $\{F_i\}_{i\in M}$ is the set $\bigcup_{i\in M} F_i$.

In our case, type $\sum_{\mathsf{i}:\mathsf{M}} \mathsf{F}\ \mathsf{i}$ will be analogous to $\bigcup_{i\in M} F_i$.

Define:

$$\mathsf{C} :\equiv \sum_{\mathsf{i}:\mathsf{M}} \mathsf{F}\ \mathsf{i}$$

Notice $\mathsf{C} : \mathcal{U}_{\max\{i,j\}}$ since $\mathsf{M} : \mathcal{U}_i$ and $\mathsf{F}\ \mathsf{i} : \mathcal{U}_j$ for every $\mathsf{i} : \mathsf{M}$ (see Remark 1.5.29).

Next, we need to build a function $\mathsf{f} : \prod_{\mathsf{i}:\mathsf{M}} (\mathsf{F}\ \mathsf{i}) \to \mathsf{C}$. Define:

$$\mathsf{f} :\equiv \lambda(\mathsf{i}:\mathsf{M})(\mathsf{j}:\mathsf{F}\ \mathsf{i}).\ (\mathsf{i}, \mathsf{j})$$

Now, we need to prove that the following type is inhabited:

$$\prod_{(\mathsf{Y},\mathsf{g}):\mathsf{Cocone}_{\max\{i,j\}}\ \mathsf{F}} \sum_{\mathsf{h}:\mathsf{C}\to\mathsf{Y}} \prod_{\mathsf{i}:\mathsf{M}} \prod_{\mathsf{w}:\mathsf{F}\ \mathsf{i}} \mathsf{g}\ \mathsf{i}\ \mathsf{w} = \mathsf{h}\ (\mathsf{f}\ \mathsf{i}\ \mathsf{w})$$

[31] Although we may use the stronger notion of *colimit* (see [5]), the notion of weak colimit will be enough for our purposes, see Remark 3.3.7.

Let $(\mathsf{Y}, \mathsf{g}) : \mathsf{Cocone}_{\max\{i,j\}}\ \mathsf{F}$. Define h by Σ-recursion (Lemma 1.5.25):

$$\mathsf{h}\ (\mathsf{i}, \mathsf{j}) :\equiv \mathsf{g}\ \mathsf{i}\ \mathsf{j}$$

which is well-typed, since $\mathsf{g}\ \mathsf{i}\ \mathsf{j} : \mathsf{Y}$.

Now, let $\mathsf{i} : \mathsf{M}$ and $\mathsf{w} : \mathsf{F}\ \mathsf{i}$. Then, we have by definition:

$$\begin{aligned}\mathsf{h}\ (\mathsf{f}\ \mathsf{i}\ \mathsf{w}) &\equiv \mathsf{h}\ (\mathsf{i}, \mathsf{w}) \\ &\equiv \mathsf{g}\ \mathsf{i}\ \mathsf{w}\end{aligned}$$

And so, $\mathsf{g}\ \mathsf{i}\ \mathsf{w} = \mathsf{h}\ (\mathsf{f}\ \mathsf{i}\ \mathsf{w})$ is inhabited.

□

Contrary to what was done in Definition 3.2.5, we do not need to specialize the concept of H-coalgebra (Definition 1.10.4) to sets. See Section 3.3.5 for a discussion on this.

Next, we need to define the concept of weakly final coalgebra. We define two versions, as was done in Definition 3.2.5. The difference between the two versions is in the universe level of the H-coalgebra quantified in the Π-type (see Definition 3.3.4 below). In type $\mathsf{LLWfinCoAlg}_k\ \mathsf{H}$, the level is $k-1$. In type $\mathsf{WfinCoAlg}_k\ \mathsf{H}$ the level is k.

Definition 3.3.4 ((Lower level) weakly final H-coalgebra)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor.

(i) Let $0 < k \leq i$ be a universe level. Any term of type:

$$\mathsf{LLWfinCoAlg}_k\ \mathsf{H} :\equiv \sum_{(\mathsf{W},\mathsf{Out}_\mathsf{W}):\mathsf{CoAlg}_k\ \mathsf{H}}\ \prod_{(\mathsf{B},\mathsf{Out}_\mathsf{B}):\mathsf{CoAlg}_{k-1}\ \mathsf{H}} \mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{W}$$

will be called a *lower level weakly final* H*-coalgebra* (*LLWfinCoAlg* for short) at level k.

(ii) Let $k \leq i$ be a universe level. Any term of type:

$$\mathsf{WfinCoAlg}_k\ \mathsf{H} :\equiv \sum_{(\mathsf{W},\mathsf{Out}_\mathsf{W}):\mathsf{CoAlg}_k\ \mathsf{H}}\ \prod_{(\mathsf{B},\mathsf{Out}_\mathsf{B}):\mathsf{CoAlg}_k\ \mathsf{H}} \mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{W}$$

will be called a *weakly final* H*-coalgebra* (*WfinCoAlg* for short) at level k.

▲

Convention 3.3.5. We will write lower level weakly final H-coalgebras as $(\mathsf{W}, \mathsf{Out}_\mathsf{W}, \mathsf{m})$, instead of $((\mathsf{W}, \mathsf{Out}_\mathsf{W}), \mathsf{m})$. The same applies to weakly final H-coalgebras.

▲

Both versions are similar to Definition 1.10.8, except that type $\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{W}$ is not asserted to be contractible. This means that in a (lower level) weakly final coalgebra, the coalgebra morphism is not required to be unique.

Also, a LLWfinCoAlg residing at level k will be weakly final *relative to* H-coalgebras at level $k-1$ (hence the words *lower level* in their name). Instead, a WfinCoAlg residing at level k will be weakly final relative to H-coalgebras at level k.

Lemma 3.3.6 will show that generalizing the Knaster-Tarski construction only produces a LLWfinCoAlg. This restriction will carry on to the next section, where we will *refine* the LLWfinCoAlg into a lower level *final* H-coalgebra. To the contrary, the existence of a WfinCoAlg for an arbitrary endofunctor can be answered in the negative (see Corollary 3.3.33 for an example of functor without a WfinCoAlg).

Nevertheless, we state both versions because the refinement process of Section 3.3.2 applies to both a LLWfinCoAlg and a WfinCoAlg (i.e. if we start with a LLWfinCoAlg we will get a lower level final H-coalgebra; if we start with a WfinCoAlg we will get a final H-coalgebra, see Definition 3.3.8 below).

Now, we state the main result on this section.

Lemma 3.3.6. *Let $i > 0$ and $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ a functor. Then, the following type is inhabited:*

$$\mathsf{LLWfinCoAlg}_i\ \mathsf{H}$$

i.e. any endofunctor has a lower level weakly final H*-coalgebra.*

Proof. First, we need to construct $(\mathsf{W}, \mathsf{Out}_\mathsf{W}) : \mathsf{CoAlg}_i\ \mathsf{H}$.

In the proof of Lemma 2.3.6(ii), the greatest fixpoint was the least upper bound of the set of all postfixpoints of the monotone function. So, we will define W to be a weak colimit for the family of H-coalgebras at level $i-1$ (remember, an H-coalgebra is a generalization to the notion of postfixpoint). Notice that we can diminish level i by one because $i > 0$.

The family of H-coalgebras at level $i-1$ is represented by the first projection function:

$$\mathsf{pr}_1 : (\mathsf{CoAlg}_{i-1}\ \mathsf{H}) \to \mathcal{U}_{i-1}$$

where type $\mathsf{CoAlg}_{i-1}\ \mathsf{H}$ is acting as the "collection of indices".

Now, $(\mathsf{CoAlg}_{i-1}\ \mathsf{H}) : \mathcal{U}_i$, because:

- $\mathcal{U}_{i-1} : \mathcal{U}_i$ by the universe introduction rule.
- For every $\mathsf{A} : \mathcal{U}_{i-1}$, type $\mathsf{A} \to (\mathsf{H}\ \mathsf{A})$ is at level $\max\{i-1, i\} = i$, since $\mathsf{A} : \mathcal{U}_{i-1}$ and $\mathsf{H}\ \mathsf{A} : \mathcal{U}_i$.
- Therefore, type $\mathsf{CoAlg}_{i-1}\ \mathsf{H} \equiv \sum_{\mathsf{A}:\mathcal{U}_{i-1}} \mathsf{A} \to (\mathsf{H}\ \mathsf{A})$ is at level $\max\{i, i\} = i$.

So, by Lemma 3.3.3, family pr_1 has a weak colimit at level $\max\{i, i-1\} = i$:

$$((\mathsf{W}, \mathsf{f}), \mathsf{q}) : \mathsf{WCoLimit}_i\ \mathsf{pr}_1 \tag{3.21}$$

Hence, W corresponds to type (see proof of Lemma 3.3.3):

$$\sum_{\mathsf{A}:\mathsf{CoAlg}_{i-1}\ \mathsf{H}} \mathsf{pr}_1\ \mathsf{A}$$

and (W, f) is a cocone over pr_1.

Now, we need to construct a function $\mathsf{Out}_\mathsf{W} : \mathsf{W} \to (\mathsf{H}\ \mathsf{W})$.

Since W is a weak colimit for pr_1, if we can construct a proof that $\mathsf{H}\ \mathsf{W}$ is a cocone over pr_1, there will be a function of type $\mathsf{W} \to (\mathsf{H}\ \mathsf{W})$.

Claim 1: $\mathsf{H}\ \mathsf{W}$ forms a cocone over pr_1.

We need to build a function of type $\prod_{\mathsf{Y}:\mathsf{CoAlg}_{i-1}\ \mathsf{H}} (\mathsf{pr}_1\ \mathsf{Y}) \to (\mathsf{H}\ \mathsf{W})$, which can be rewritten as:

$$\prod_{(\mathsf{A},\mathsf{Out}_\mathsf{A}):\mathsf{CoAlg}_{i-1}\ \mathsf{H}} \mathsf{A} \to (\mathsf{H}\ \mathsf{W})$$

by Convention 1.5.27.

So, let $(\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{CoAlg}_{i-1}\ \mathsf{H}$. Consider the composition:

$$\mathsf{A} \xrightarrow{\mathsf{Out}_\mathsf{A}} \mathsf{H}\ \mathsf{A} \xrightarrow{\mathsf{map}^\mathsf{H}\ (\mathsf{f}\ (\mathsf{A},\mathsf{Out}_\mathsf{A}))} \mathsf{H}\ \mathsf{W}$$

where function $\mathsf{f}\ (\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{A} \to \mathsf{W}$ exists, because (W, f) is a cocone over pr_1.

This proves the claim.

Therefore, since W is a weak colimit, by Claim 1 there is a function $\mathsf{Out}_\mathsf{W} : \mathsf{W} \to (\mathsf{H}\ \mathsf{W})$ satisfying:

$$\prod_{(\mathsf{A},\mathsf{Out}_\mathsf{A}):\mathsf{CoAlg}_{i-1}\ \mathsf{H}} \prod_{\mathsf{y}:\mathsf{A}} \mathsf{map}^\mathsf{H}\ (\mathsf{f}\ (\mathsf{A}, \mathsf{Out}_\mathsf{A}))\ (\mathsf{Out}_\mathsf{A}\ \mathsf{y}) = \mathsf{Out}_\mathsf{W}\ (\mathsf{f}\ (\mathsf{A}, \mathsf{Out}_\mathsf{A})\ \mathsf{y}) \tag{3.22}$$

We have constructed $(\mathsf{W}, \mathsf{Out}_\mathsf{W}) : \mathsf{CoAlg}_i\ \mathsf{H}$. It remains to show that the following type is inhabited:

$$\prod_{(\mathsf{B},\mathsf{Out}_\mathsf{B}):\mathsf{CoAlg}_{i-1}\ \mathsf{H}} \mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{W} \tag{3.23}$$

Let $(\mathsf{B}, \mathsf{Out}_\mathsf{B}) : \mathsf{CoAlg}_{i-1}\ \mathsf{H}$. We have to construct an H-coalgebra morphism from $(\mathsf{B}, \mathsf{Out}_\mathsf{B})$ to $(\mathsf{W}, \mathsf{Out}_\mathsf{W})$.

But (W, f) is a cocone over pr_1 and $(\mathsf{B}, \mathsf{Out}_\mathsf{B})$ is in the family, which means we have:

$$\mathsf{f}\ (\mathsf{B}, \mathsf{Out}_\mathsf{B}) : (\mathsf{pr}_1\ (\mathsf{B}, \mathsf{Out}_\mathsf{B})) \to \mathsf{W}$$

or equivalently, $\mathsf{f}\ (\mathsf{B}, \mathsf{Out}_\mathsf{B}) : \mathsf{B} \to \mathsf{W}$.

To show that $\mathsf{f}\ (\mathsf{B}, \mathsf{Out}_\mathsf{B})$ is a morphism, we need to prove that the following diagram commutes:

$$\begin{array}{ccc} \mathsf{B} & \xrightarrow{\mathsf{f}\ (\mathsf{B},\mathsf{Out}_\mathsf{B})} & \mathsf{W} \\ {\scriptstyle \mathsf{Out}_\mathsf{B}}\downarrow & & \downarrow{\scriptstyle \mathsf{Out}_\mathsf{W}} \\ \mathsf{H}\ \mathsf{B} & \xrightarrow[\mathsf{map}^\mathsf{H}\ (\mathsf{f}\ (\mathsf{B},\mathsf{Out}_\mathsf{B}))]{} & \mathsf{H}\ \mathsf{W} \end{array}$$

Let $\mathsf{y} : \mathsf{B}$, or equivalently, $\mathsf{y} : \mathsf{pr}_1\ (\mathsf{B}, \mathsf{Out}_\mathsf{B})$. But this diagram is type (3.22) instantiated with $(\mathsf{B}, \mathsf{In}_\mathsf{B})$ and y. □

Before ending this section, we have a remark regarding Lemma 3.3.6. This remark is similar to Remark 3.2.8.

Remark 3.3.7. Similar comments apply as in Remark 3.2.8. This time, strengthening (3.21) from a weak colimit into a colimit *seems* to be useless, because a colimit will construct unique functions *out* of type W, while type (3.23) will require to construct a unique function *into* type W, if we want a final coalgebra.

In spite of this, there is a way to construct a (lower level) final H-coalgebra out of a (lower level) weakly final H-coalgebra. This is the subject of the following section. ▲

3.3.2 Refining into a (lower level) final coalgebra

The previous section showed that any endofunctor has a LLWfinCoAlg. This section will explore if the result can be strengthened to a final H-coalgebra.

In particular, this section will show that if we are given a LLWfinCoAlg for some functor H, then we can refine it into a lower level final H-coalgebra. But also, if we are given a WfinCoAlg for some functor H, then we can refine it into a final H-coalgebra.

First, we need to define the notions of "final coalgebra" this section will use.

Definition 3.3.8 ((Lower level) final H-coalgebra)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor.

(i) Let $0 < k \leq i$ be a universe level. Any term of type:

$$\mathsf{LLFinCoAlg}_k\ \mathsf{H} :\equiv \sum_{(\mathsf{W},\mathsf{Out}_\mathsf{W}):\mathsf{CoAlg}_k\ \mathsf{H}} \prod_{(\mathsf{B},\mathsf{Out}_\mathsf{B}):\mathsf{CoAlg}_{k-1}\ \mathsf{H}} \mathsf{IsContr}\ (\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{W})$$

will be called a *lower level final* H*-coalgebra* (*LLFinCoAlg* for short) at level k.

(ii) Let $k \leq i$ be a universe level. Any term of type:

$$\mathsf{FinCoAlg}_k\ \mathsf{H} :\equiv \sum_{(\mathsf{W},\mathsf{Out}_\mathsf{W}):\mathsf{CoAlg}_k\ \mathsf{H}} \prod_{(\mathsf{B},\mathsf{Out}_\mathsf{B}):\mathsf{CoAlg}_k\ \mathsf{H}} \mathsf{IsContr}\ (\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{W})$$

will be called a *final* H*-coalgebra* (*FinCoAlg* for short) at level k. Notice this is just Definition 1.10.8, but we repeat it here for convenience. ▲

Convention 3.3.9. We will write lower level final H-coalgebras as $(\mathsf{W}, \mathsf{Out}_\mathsf{W}, \mathsf{mc})$ instead of $((\mathsf{W}, \mathsf{Out}_\mathsf{W}), \mathsf{mc})$. The same applies to final H-coalgebras. ▲

Definition 3.3.8 is like Definition 3.3.4, with the exception that coalgebra morphisms are asserted to be *unique*, i.e. type $\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{W}$ is contractible (see Definition 1.8.1).

It is obvious that a FinCoAlg is a LLFinCoAlg (just by unfolding definitions), but the other way around is not necessarily true. Although both concepts look superficially the same, LLFinCoAlgs are *weaker* than FinCoAlgs. A LLFinCoAlg is not necessarily a "fixpoint" for its functor (see Corollary 3.3.34), but any FinCoAlg is a fixpoint for its functor, as the following lemma shows.[32]

[32] This is dual to Lemma 3.2.11.

Lemma 3.3.10 (Lambek)**.** *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor. Let* $k \leq i$ *be a universe level, and* $(\mathsf{A}, \mathsf{Out}_\mathsf{A}, \mathsf{mc}) : \mathsf{FinCoAlg}_k\ \mathsf{H}$ *a final* H*-coalgebra. Then,* $\mathsf{H}\ \mathsf{A}$ *and* A *are equivalent types. In other words, the following type is inhabited:*

$$\mathsf{H}\ \mathsf{A} \simeq \mathsf{A}$$

Proof. By Lemma 1.6.18, to prove that $\mathsf{H}\ \mathsf{A}$ and A are equivalent, it is enough to construct two mutually inverse functions $(\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$ and $\mathsf{A} \to (\mathsf{H}\ \mathsf{A})$.

We take $\mathsf{Out}_\mathsf{A} : \mathsf{A} \to (\mathsf{H}\ \mathsf{A})$ as the second function. It remains to define the first function.

To make $\mathsf{H}\ \mathsf{A}$ an H-coalgebra, we require an "Out" function, consider:

$$\mathsf{map}^\mathsf{H}\ \mathsf{Out}_\mathsf{A} : (\mathsf{H}\ \mathsf{A}) \to (\mathsf{H}\ (\mathsf{H}\ \mathsf{A}))$$

Therefore, we can form the H-coalgebra $(\mathsf{H}\ \mathsf{A}, \mathsf{map}^\mathsf{H}\ \mathsf{Out}_\mathsf{A})$ at level i.

This means we can apply term mc on coalgebra $(\mathsf{H}\ \mathsf{A}, \mathsf{map}^\mathsf{H}\ \mathsf{Out}_\mathsf{A})$ to get a morphism $(\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$. More specifically, we have a morphism:

$$\mathsf{h_m} :\equiv \mathsf{center}\ (\mathsf{mc}\ (\mathsf{H}\ \mathsf{A}, \mathsf{map}^\mathsf{H}\ \mathsf{Out}_\mathsf{A})) : \mathsf{CoAlgMor}\ (\mathsf{H}\ \mathsf{A})\ \mathsf{A}$$

Therefore, we can define our second function as:

$$\mathsf{u} :\equiv \mathsf{pr}_1\ \mathsf{h_m} : (\mathsf{H}\ \mathsf{A}) \to \mathsf{A}$$

Notice that we also have a proof $(\mathsf{pr}_2\ \mathsf{h_m})$ stating that the following diagram commutes:

$$\begin{array}{ccc} \mathsf{H}\ \mathsf{A} & \xrightarrow{\mathsf{u}} & \mathsf{A} \\ {\scriptstyle \mathsf{map}^\mathsf{H}\ \mathsf{Out}_\mathsf{A}}\downarrow & & \downarrow{\scriptstyle \mathsf{Out}_\mathsf{A}} \\ \mathsf{H}\ (\mathsf{H}\ \mathsf{A}) & \xrightarrow[\mathsf{map}^\mathsf{H}\ \mathsf{u}]{} & \mathsf{H}\ \mathsf{A} \end{array} \tag{3.24}$$

Now, it remains to show that Out_A and u are inverses of each other.

- Type $\prod_{\mathsf{w}:\mathsf{A}} \mathsf{u}\ (\mathsf{Out}_\mathsf{A}\ \mathsf{w}) = \mathsf{w}$ is inhabited.

  We have the two commutative squares:

$$\begin{array}{ccccc} \mathsf{A} & \xrightarrow{\mathsf{Out}_\mathsf{A}} & \mathsf{H}\ \mathsf{A} & \xrightarrow{\mathsf{u}} & \mathsf{A} \\ {\scriptstyle \mathsf{Out}_\mathsf{A}}\downarrow & & \downarrow{\scriptstyle \mathsf{map}^\mathsf{H}\ \mathsf{Out}_\mathsf{A}} & & \downarrow{\scriptstyle \mathsf{Out}_\mathsf{A}} \\ \mathsf{H}\ \mathsf{A} & \xrightarrow[\mathsf{map}^\mathsf{H}\ \mathsf{Out}_\mathsf{A}]{} & \mathsf{H}\ (\mathsf{H}\ \mathsf{A}) & \xrightarrow[\mathsf{map}^\mathsf{H}\ \mathsf{u}]{} & \mathsf{H}\ \mathsf{A} \end{array}$$

  where the square on the right is Diagram (3.24), and the square on the left is the composition:

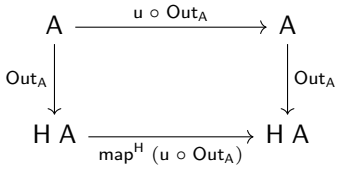
$$\mathsf{A} \xrightarrow{\mathsf{Out}_\mathsf{A}} \mathsf{H}\ \mathsf{A} \xrightarrow{\mathsf{map}^\mathsf{H}\ \mathsf{Out}_\mathsf{A}} \mathsf{H}\ (\mathsf{H}\ \mathsf{A})$$

  written twice on the sides of the square (hence the left square trivially commutes).

  This means that the "external" square commutes:

$$\begin{array}{ccc} \mathsf{A} & \xrightarrow{\mathsf{u} \circ \mathsf{Out}_\mathsf{A}} & \mathsf{A} \\ {\scriptstyle \mathsf{Out}_\mathsf{A}}\downarrow & & \downarrow{\scriptstyle \mathsf{Out}_\mathsf{A}} \\ \mathsf{H}\ \mathsf{A} & \xrightarrow[(\mathsf{map}^\mathsf{H}\ \mathsf{u}) \circ (\mathsf{map}^\mathsf{H}\ \mathsf{Out}_\mathsf{A})]{} & \mathsf{H}\ \mathsf{A} \end{array}$$

which, by functorial mapping of H (see Definition 1.9.6), is equivalent to the commutative diagram:



This last diagram claims that function $\mathsf{u} \circ \mathsf{Out_A}$ is a coalgebra morphism from A to A. But the identity function $\mathsf{id_A}$ is also a coalgebra morphism from A to A (Lemma 1.10.7). Since A is a final H-coalgebra, we must have that $\mathsf{u} \circ \mathsf{Out_A} = \mathsf{id_A}$ holds, because morphisms from A to A are unique.

- Type $\prod_{\mathsf{w:H\ A}} \mathsf{Out_A\ (u\ w)} = \mathsf{w}$ is inhabited.
  Let $\mathsf{w} : \mathsf{H\ A}$. We have the following sequence of identifications:

$$\begin{aligned}
\mathsf{Out_A\ (u\ w)} &\overset{(1)}{=} \mathsf{map^H\ u\ (map^H\ Out_A\ w)} \\
&\overset{(2)}{=} \mathsf{map^H\ (u \circ Out_A)\ w} \\
&\overset{(3)}{=} \mathsf{map^H\ id_A\ w} \\
&\overset{(4)}{=} \mathsf{id_{(H\ A)} w} \\
&\overset{(5)}{=} \mathsf{w}
\end{aligned}$$

  where (1) follows by commutativity of Diagram (3.24), (2) by functorial mapping of H, (3) by the previous case and function extensionality, (4) by functorial mapping of H on the identity function, and (5) by definition of the identity function.

□

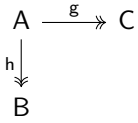
If we try to repeat Lambek's Lemma for a LLFinCoAlg $(\mathsf{A}, \mathsf{Out_A}, \mathsf{mc})$, we will get stuck while defining function $(\mathsf{H\ A}) \to \mathsf{A}$. The reason is that type $\mathsf{H\ A}$ is at level i. Therefore, we will not be able to apply mc on $\mathsf{H\ A}$ because mc will expect a coalgebra at level $i-1$. Later, we will construct a LLFinCoAlg for which the conclusion of Lambek's Lemma does not hold (see Corollary 3.3.34).

Before starting the refinement process, we need a couple of lemmas.

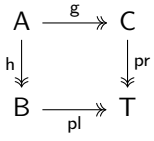
Lemma 3.3.11 (Coalgebra pushout square)**.** *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor,* $k, l, m \leq i$ *universe levels, and* $(\mathsf{A}, \mathsf{Out_A}) : \mathsf{CoAlg}_k\ \mathsf{H}$*,* $(\mathsf{B}, \mathsf{Out_B}) : \mathsf{CoAlg}_l\ \mathsf{H}$*, and* $(\mathsf{C}, \mathsf{Out_C}) : \mathsf{CoAlg}_m\ \mathsf{H}$ *three* H*-coalgebras.*

If $\mathsf{h} : \mathsf{A} \to \mathsf{B}$ *and* $\mathsf{g} : \mathsf{A} \to \mathsf{C}$ *are two surjective*[33] *coalgebra morphisms, then there is an* H*-coalgebra* $(\mathsf{T}, \mathsf{Out_T}) : \mathsf{CoAlg}_{\max\{k,l,m\}}\ \mathsf{H}$ *and two surjective coalgebra morphisms* $\mathsf{pl} : \mathsf{B} \to \mathsf{T}$*,* $\mathsf{pr} : \mathsf{C} \to \mathsf{T}$ *such that* $(\mathsf{pl} \circ \mathsf{h}) \sim (\mathsf{pr} \circ \mathsf{g})$*.*

In other words, given a diagram of coalgebras and surjective coalgebra morphisms:[34]



this diagram can be completed into a commutative "pushout square" of coalgebras and surjective coalgebra morphisms:[35]



[33] See Definition 1.11.9.

[34] From now on, surjective functions will be denoted by double-headed arrows (↠) in diagrams.

[35] It is possible to prove the stronger result that coalgebra $(\mathsf{T}, \mathsf{Out_T})$ satisfies the universal property for pushouts, but it will not be needed on this report.

Proof. Define:[36]

$$\mathsf{T} :\equiv \mathsf{Push}\ \mathsf{h}\ \mathsf{g} : \mathcal{U}_{\max\{k,l,m\}}$$

We have two functions:

$$\mathsf{h}_1 :\equiv ((\mathsf{map}^{\mathsf{H}}\ \mathsf{pushl}) \circ \mathsf{Out}_{\mathsf{B}}) : \mathsf{B} \to (\mathsf{H}\ \mathsf{T})$$
$$\mathsf{h}_2 :\equiv ((\mathsf{map}^{\mathsf{H}}\ \mathsf{pushr}) \circ \mathsf{Out}_{\mathsf{C}}) : \mathsf{C} \to (\mathsf{H}\ \mathsf{T})$$

Therefore, if we are able to prove $\prod_{\mathsf{a}:\mathsf{A}} \mathsf{h}_1\ (\mathsf{h}\ \mathsf{a}) = \mathsf{h}_2\ (\mathsf{g}\ \mathsf{a})$, by the recursion principle for pushouts (Lemma 1.11.22) we will have a function $\mathsf{Out}_{\mathsf{T}} : \mathsf{T} \to (\mathsf{H}\ \mathsf{T})$ satisfying:

$$\mathsf{Out}_{\mathsf{T}}\ (\mathsf{pushl}\ \mathsf{b}) \equiv \mathsf{h}_1\ \mathsf{b} \equiv \mathsf{map}^{\mathsf{H}}\ \mathsf{pushl}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{b})$$
$$\mathsf{Out}_{\mathsf{T}}\ (\mathsf{pushr}\ \mathsf{c}) \equiv \mathsf{h}_2\ \mathsf{c} \equiv \mathsf{map}^{\mathsf{H}}\ \mathsf{pushr}\ (\mathsf{Out}_{\mathsf{C}}\ \mathsf{c})$$

for any $\mathsf{b} : \mathsf{B}$ and $\mathsf{c} : \mathsf{C}$.

So, we need to prove that $\prod_{\mathsf{a}:\mathsf{A}} \mathsf{h}_1\ (\mathsf{h}\ \mathsf{a}) = \mathsf{h}_2\ (\mathsf{g}\ \mathsf{a})$ is inhabited, or equivalently:

$$\prod_{\mathsf{a}:\mathsf{A}} \mathsf{map}^{\mathsf{H}}\ \mathsf{pushl}\ (\mathsf{Out}_{\mathsf{B}}\ (\mathsf{h}\ \mathsf{a})) = \mathsf{map}^{\mathsf{H}}\ \mathsf{pushr}\ (\mathsf{Out}_{\mathsf{C}}\ (\mathsf{g}\ \mathsf{a}))$$

Let $\mathsf{a} : \mathsf{A}$. We have:

$$\begin{aligned}
\mathsf{map}^{\mathsf{H}}\ \mathsf{pushl}\ (\mathsf{Out}_{\mathsf{B}}\ (\mathsf{h}\ \mathsf{a})) &\overset{(1)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{pushl}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{h}\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a})) \\
&\overset{(2)}{=} \mathsf{map}^{\mathsf{H}}\ (\mathsf{pushl} \circ \mathsf{h})\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a}) \\
&\overset{(3)}{=} \mathsf{map}^{\mathsf{H}}\ (\mathsf{pushr} \circ \mathsf{g})\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a}) \\
&\overset{(4)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{pushr}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{g}\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a})) \\
&\overset{(5)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{pushr}\ (\mathsf{Out}_{\mathsf{C}}\ (\mathsf{g}\ \mathsf{a}))
\end{aligned}$$

where (1) holds because h is a coalgebra morphism, (2) follows by functorial mapping of H, (3) follows by the higher constructor $\mathsf{pushiden}$ for $\mathsf{Push}\ \mathsf{h}\ \mathsf{g}$ (see Lemma 1.11.20) and function extensionality, (4) follows by functorial mapping of H, and (5) holds because g is a coalgebra morphism.

Therefore, function $\mathsf{Out}_{\mathsf{T}}$ exists. Now, define:

$$\mathsf{pl} :\equiv \mathsf{pushl}$$
$$\mathsf{pr} :\equiv \mathsf{pushr}$$

Functions $\mathsf{pl} : \mathsf{B} \to \mathsf{T}$ and $\mathsf{pr} : \mathsf{C} \to \mathsf{T}$ are coalgebra morphisms, since we have by definition of T:

$$\mathsf{Out}_{\mathsf{T}}\ (\mathsf{pl}\ \mathsf{b}) \equiv \mathsf{Out}_{\mathsf{T}}\ (\mathsf{pushl}\ \mathsf{b}) \equiv \mathsf{map}^{\mathsf{H}}\ \mathsf{pushl}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{b}) \equiv \mathsf{map}^{\mathsf{H}}\ \mathsf{pl}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{b})$$
$$\mathsf{Out}_{\mathsf{T}}\ (\mathsf{pr}\ \mathsf{c}) \equiv \mathsf{Out}_{\mathsf{T}}\ (\mathsf{pushr}\ \mathsf{c}) \equiv \mathsf{map}^{\mathsf{H}}\ \mathsf{pushr}\ (\mathsf{Out}_{\mathsf{C}}\ \mathsf{c}) \equiv \mathsf{map}^{\mathsf{H}}\ \mathsf{pr}\ (\mathsf{Out}_{\mathsf{C}}\ \mathsf{c})$$

for any $\mathsf{b} : \mathsf{B}$ and $\mathsf{c} : \mathsf{C}$.

Since functions h and g are surjective functions, we have that functions pl and pr are surjective functions, by Lemma 1.11.23.

The pushout square commutes by the higher constructor $\mathsf{pushiden}$ for $\mathsf{Push}\ \mathsf{h}\ \mathsf{g}$ (see Lemma 1.11.20).

□

Lemma 3.3.12 (Coalgebra coequalizer diagram)**.** *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor,* $k, m \leq i$ *universe levels, and* $(\mathsf{A}, \mathsf{Out}_{\mathsf{A}}) : \mathsf{CoAlg}_k\ \mathsf{H}$, $(\mathsf{B}, \mathsf{Out}_{\mathsf{B}}) : \mathsf{CoAlg}_m\ \mathsf{H}$ *two* H*-coalgebras.*

If $\mathsf{f} : \mathsf{A} \to \mathsf{B}$ *and* $\mathsf{g} : \mathsf{A} \to \mathsf{B}$ *are two coalgebra morphisms, then there is an* H*-coalgebra* $(\mathsf{T}, \mathsf{Out}_{\mathsf{T}}) : \mathsf{CoAlg}_{\max\{k,m\}}\ \mathsf{H}$ *and a surjective coalgebra morphism* $\mathsf{p} : \mathsf{B} \to \mathsf{T}$ *such that* $(\mathsf{p} \circ \mathsf{f}) \sim (\mathsf{p} \circ \mathsf{g})$.

[36] See Lemma 1.11.20.

In other words, given a diagram of coalgebras and coalgebra morphisms:

$$\mathsf{A} \underset{\mathsf{g}}{\overset{\mathsf{f}}{\rightrightarrows}} \mathsf{B}$$

this diagram can be completed into a commutative "coequalizer diagram" of coalgebras and coalgebra morphisms:[37]

$$\mathsf{A} \underset{\mathsf{g}}{\overset{\mathsf{f}}{\rightrightarrows}} \mathsf{B} \xrightarrow{\mathsf{p}} \!\!\!\!\rightarrow \mathsf{T}$$

Proof. Define:[38]

$$\mathsf{T} :\equiv \mathsf{Coeq}\ \mathsf{f}\ \mathsf{g} : \mathcal{U}_{\max\{k,m\}}$$

We have a function:

$$\mathsf{h} :\equiv ((\mathsf{map}^{\mathsf{H}}\ \mathsf{coeq}) \circ \mathsf{Out}_{\mathsf{B}}) : \mathsf{B} \to (\mathsf{H}\ \mathsf{T})$$

Therefore, if we are able to prove $\prod_{\mathsf{a}:\mathsf{A}} \mathsf{h}\ (\mathsf{f}\ \mathsf{a}) = \mathsf{h}\ (\mathsf{g}\ \mathsf{a})$, by the recursion principle for coequalizers (Lemma 1.11.18) we will have a function $\mathsf{Out}_{\mathsf{T}} : \mathsf{T} \to (\mathsf{H}\ \mathsf{T})$ satisfying:

$$\mathsf{Out}_{\mathsf{T}}\ (\mathsf{coeq}\ \mathsf{b}) \equiv \mathsf{h}\ \mathsf{b} \equiv \mathsf{map}^{\mathsf{H}}\ \mathsf{coeq}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{b})$$

for any $\mathsf{b} : \mathsf{B}$.

So, we need to prove that $\prod_{\mathsf{a}:\mathsf{A}} \mathsf{h}\ (\mathsf{f}\ \mathsf{a}) = \mathsf{h}\ (\mathsf{g}\ \mathsf{a})$ is inhabited, or equivalently:

$$\prod_{\mathsf{a}:\mathsf{A}} \mathsf{map}^{\mathsf{H}}\ \mathsf{coeq}\ (\mathsf{Out}_{\mathsf{B}}\ (\mathsf{f}\ \mathsf{a})) = \mathsf{map}^{\mathsf{H}}\ \mathsf{coeq}\ (\mathsf{Out}_{\mathsf{B}}\ (\mathsf{g}\ \mathsf{a}))$$

Let $\mathsf{a} : \mathsf{A}$. We have:

$$\begin{aligned}
\mathsf{map}^{\mathsf{H}}\ \mathsf{coeq}\ (\mathsf{Out}_{\mathsf{B}}\ (\mathsf{f}\ \mathsf{a})) &\overset{(1)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{coeq}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{f}\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a})) \\
&\overset{(2)}{=} \mathsf{map}^{\mathsf{H}}\ (\mathsf{coeq} \circ \mathsf{f})\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a}) \\
&\overset{(3)}{=} \mathsf{map}^{\mathsf{H}}\ (\mathsf{coeq} \circ \mathsf{g})\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a}) \\
&\overset{(4)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{coeq}\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{g}\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a})) \\
&\overset{(5)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{coeq}\ (\mathsf{Out}_{\mathsf{B}}\ (\mathsf{g}\ \mathsf{a}))
\end{aligned}$$

where (1) holds because f is a coalgebra morphism, (2) follows by functorial mapping of H, (3) follows by the higher constructor $\mathsf{coeqiden}$ for $\mathsf{Coeq}\ \mathsf{f}\ \mathsf{g}$ (see Definition 1.11.15) and function extensionality, (4) follows by functorial mapping of H, and (5) holds because g is a coalgebra morphism.

Therefore, function $\mathsf{Out}_{\mathsf{T}}$ exists. Now, define:

$$\mathsf{p} :\equiv \mathsf{coeq}$$

Function $\mathsf{p} : \mathsf{B} \to \mathsf{T}$ is a coalgebra morphism, since we have by definition of T:

$$\mathsf{Out}_{\mathsf{T}}\ (\mathsf{p}\ \mathsf{b}) \equiv \mathsf{Out}_{\mathsf{T}}\ (\mathsf{coeq}\ \mathsf{b}) \equiv \mathsf{map}^{\mathsf{H}}\ \mathsf{coeq}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{b}) \equiv \mathsf{map}^{\mathsf{H}}\ \mathsf{p}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{b})$$

for any $\mathsf{b} : \mathsf{B}$.

By Lemma 1.11.19, function p is a surjective function because it is the constructor for the coequalizer.

The coequalizer diagram commutes by the higher constructor $\mathsf{coeqiden}$ for $\mathsf{Coeq}\ \mathsf{f}\ \mathsf{g}$ (see Definition 1.11.15).

□

Given a LLWfinCoAlg (or a WfinCoAlg) A, the refinement process into a LLFinCoAlg (FinCoAlg respectively) will consist on the following steps:

[37] It is possible to prove the stronger result that coalgebra $(\mathsf{T}, \mathsf{Out}_{\mathsf{T}})$ satisfies the universal property for coequalizers, but it will not be needed on this report.

[38] See Definition 1.11.15.

1. Identify those terms in A that "behave the same way". Denote the resultant type $\mathsf{A_F}$.
2. Prove that $\mathsf{A_F}$ is an H-coalgebra.
3. Construct a coinduction principle for $\mathsf{A_F}$.
4. Use the coinduction principle to prove the finality of $\mathsf{A_F}$.

The last step is the only one where we need A to be a LLWfinCoAlg (or a WfinCoAlg). All other steps can be performed with the weaker assumption that A is an H-coalgebra.

Step 1: Identify terms that behave the same way

Let $(\mathsf{A}, \mathsf{Out_A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ be a coalgebra, and $\mathsf{w}, \mathsf{y} : \mathsf{A}$ two terms. Many binary relations $\approx : \mathsf{A} \to \mathsf{A} \to \mathbf{Prop}_i$ can be defined for expressing that terms w and y "behave the same way" (see [19]). The intention on defining such "behavior" relation is that if we take the quotient of coalgebra A under the relation, we will be able to treat two elements of A as the same if they behave the same way. On this report we will be interested on two "behavior" relations: *behavioral equivalence* and *bisimilarity*.

To motivate both concepts, let us work for a moment in standard mathematics. Let $\mathtt{SET}$ be the category of sets, and let us suppose we have a set A. Define the functor[39] $H_A : \mathtt{SET} \to \mathtt{SET}$ as:

$$H_A(X) = A \times X$$
$$H_A(f)(a, x) = (a, f(x))$$

where X is a set and $f : C \to D$ a function between sets C and D.

Let A^ω be the set of countably infinite sequences on elements of A (A^ω is also called the set of *streams* on A). We denote elements of A^ω as $(a_1, a_2, a_3, \ldots)$, where $a_1, a_2, a_3, \ldots \in A$.

Let us define the $head : A^\omega \to A$ and $tail : A^\omega \to A^\omega$ functions as follows:

$$head(a_1, a_2, a_3, \ldots) = a_1$$
$$tail(a_1, a_2, a_3, \ldots) = (a_2, a_3, a_4, \ldots)$$

This way, we can form an H_A-coalgebra[40] on A^ω, by defining $Out_{A^\omega} : A^\omega \to H_A(A^\omega)$ as follows:

$$Out_{A^\omega}(s) = (head(s), tail(s))$$

We can consider elements of A^ω as sequences of *observations*: given $s \in A^\omega$, every time we apply $head(s)$, we get the *current available observation* in s; and every time we apply $tail(s)$, we change s into its *next state*. Therefore, to get the second available observation, apply $head(tail(s))$; to get the third available observation, apply $head(tail(tail(s)))$; and so on.

Given $s, s' \in A^\omega$, what does it mean for s and s' to "behave the same way"? Our intuition tells us that they behave the same way if we cannot tell them apart from the point of view of observations, i.e. both match on their first observation, on their second observation, on their third observation, etc.

So, we say that s and s' are *behaviorally equivalent* if $head(tail^n(s)) = head(tail^n(s'))$ for every $n \geq 0$, where $tail^n$ is defined recursively as follows:

$$tail^0(x) = x$$
$$tail^{n+1}(x) = tail^n(tail(x))$$

i.e. s and s' are behaviorally equivalent if they match on all their observations.

We can express this more succinctly by defining the *behavior* function $beh : A^\omega \to A^\omega$, which maps each $x \in A^\omega$ into its *observation history*:

$$beh(x) = (head(tail^0(x)),\ head(tail^1(x)),\ head(tail^2(x)),\ \ldots)$$

[39] See Definition 1.9.2.
[40] See Definition 1.10.1.

so that s and s' are behaviorally equivalent if and only if $beh(s) = beh(s')$.

The behavior function is a coalgebra morphism,[41] since we have for any $x \in A^\omega$:

$$\begin{aligned}
H_A(beh)(Out_{A^\omega}(x)) &= H_A(beh)(head(x), tail(x)) \\
&= (head(x), beh(tail(x))) \\
&= (head(x),\ (head(tail^0(tail(x))),\ head(tail^1(tail(x))), \\
&\qquad\qquad head(tail^2(tail(x))),\ \ldots)\ ) \\
&= (head(x),\ (head(tail^1(x)),\ head(tail^2(x)), \\
&\qquad\qquad head(tail^3(x)),\ \ldots)\ ) \\
&= (head(tail^0(x)),\ (head(tail^1(x)),\ head(tail^2(x)), \\
&\qquad\qquad\qquad head(tail^3(x)),\ \ldots)\ ) \\
&= (head(beh(x)), tail(beh(x))) \\
&= Out_{A^\omega}(beh(x))
\end{aligned}$$

In fact, given $s, s' \in A^\omega$, we have the following property: "Streams s and s' are behaviorally equivalent *if and only if* there is a coalgebra (T, Out_T) and two coalgebra morphisms $f, g : A^\omega \to T$ such that $f(s) = g(s')$".

For the $\Rightarrow$ direction, just define T to be A^ω, and define f and g to be the behavior function beh (which is a coalgebra morphism). The proof for the $\Leftarrow$ direction can be found at Example 2.6 in [11].

The above property is expressing: "To check that s and s' are behaviorally equivalent, it is enough to find a coalgebra (T, Out_T) encoding *observation histories*, and two coalgebra morphisms $f, g : A^w \to T$ acting as *behavior functions*, in such a way that the observation histories of s and s' are exactly the same (i.e. $f(s) = g(s')$)".

This property allows the generalization of "behavioral equivalence" to arbitrary coalgebras, as we state in the following candidate definition for behavioral equivalence. For the moment, we will express this candidate definition in standard mathematics. Once we have our final definition, we will formalize it in HoTT.

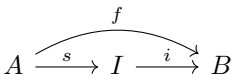
Candidate definition 3.3.13 (Behavioral equivalence, version 1). Let $H : \texttt{SET} \to \texttt{SET}$ be a functor. Let (A, Out_A) be an H-coalgebra and $x, y \in A$. We say that x and y are behaviorally equivalent if there is an H-coalgebra (T, Out_T) and two coalgebra morphisms $f, g : A \to T$ such that $f(x) = g(y)$.

▲

In the literature, the concept of behavioral equivalence is defined for two arbitrary coalgebras, and not just for one coalgebra as we did in the above candidate definition (see Definition 2.7 in [11], and Definition 163 in Chapter 6 of [7]). But this simplified version will be enough for the purposes on this report.

However, there is a small issue with version 1 of behavioral equivalence. The author took the idea of using behavioral equivalence from [11], because, as shown in there, taking a quotient over behavioral equivalence produces a coalgebra again. However, if we follow the results in [11], at some point we will need to provide a *non-constructive* proof for the following statement:

"Let (A, Out_A) and (B, Out_B) be two coalgebras (where $A \neq \emptyset$) and $f : A \to B$ a coalgebra morphism. Then, f can be factorized into a surjective coalgebra morphism $s : A \to I$, followed by an injective coalgebra morphism $i : I \to B$:



where (I, Out_I) is some coalgebra".

Although we will not do the details,[42] the proof for this statement defines I to be the image $f[A]$, function s to be the codomain restriction of f to $f[A]$, and i to be the inclusion function $f[A] \to B$. Then, the proof asserts that the inclusion function i has a left inverse, and then proceeds to define function Out_I by using the left inverse of i. In classical mathematics, it is true that inclusion functions with non-empty domain have left inverses. However, this is not necessarily true in constructive mathematics, because defining a left inverse for an arbitrary inclusion function requires the law of excluded middle.

[41] See Definition 1.10.2.

[42] Details for the proof can be found at Lemma 3.5 in [11].

So, if we want to avoid excluded middle, we need to avoid using the above statement in our development. One way to accomplish this is to make sure that all coalgebra morphisms in our argument are always surjective. This way, we will not need to factorize them through a surjective function (since they are already surjective functions). It turns out that the only thing we need to do to guarantee this "surjective coalgebra morphism property" is to change the candidate definition for behavioral equivalence as follows.

Candidate definition 3.3.14 (Behavioral equivalence, version 2). Let $H : \mathtt{SET} \to \mathtt{SET}$ be a functor. Let (A, Out_A) be an H-coalgebra and $x, y \in A$. We say that x and y are behaviorally equivalent if there is an H-coalgebra (T, Out_T) and two *surjective* coalgebra morphisms $f, g : A \to T$ such that $f(x) = g(y)$.

▲

In other words, we will require both morphisms f, g to be surjective.

It turns out that, *classically*, the two versions for behavioral equivalence are logically equivalent. It is obvious that version 2 implies version 1. However, to prove version 2 from version 1 we need to use excluded middle. We omit the proof as it is not critical for our development.

We can now express behavioral equivalence (version 2) in HoTT.

Definition 3.3.15 (Pure behavioral equivalence)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k, l \leq i$ universe levels, and $(\mathsf{A}, \mathsf{Out_A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ an H-coalgebra.

Let $\mathsf{w}, \mathsf{y} : \mathsf{A}$ be two terms. We define type:

$$\mathsf{PurelyBehEquiv}_l\ \mathsf{w}\ \mathsf{y} :\equiv \sum_{(\mathsf{T},\mathsf{Out_T}):\mathsf{CoAlg}_l\ \mathsf{H}} \sum_{(\mathsf{f},\mathsf{mf}),(\mathsf{g},\mathsf{mg}):\mathsf{CoAlgMor\ A\ T}} (\mathsf{IsSurjection\ f}) \times (\mathsf{IsSurjection\ g}) \times (\mathsf{f\ w} = \mathsf{g\ y})$$

If type $(\mathsf{PurelyBehEquiv}_l\ \mathsf{w}\ \mathsf{y})$ is inhabited, we say that w and y are *purely behaviorally equivalent.*[43]

▲

A second concept that also captures an idea of "similar behavior" is *bisimilarity*. Behavioral equivalence expresses that x and y have exactly the same *global* observation history, while bisimilarity states that x and y observationally imitate each other, but *locally*. To make this statement clear, let us work with H_A-coalgebra $(A^\omega, Out_{A^\omega})$, as we did with behavioral equivalence.

Let $s, s' \in A^\omega$ be two streams. After we observe the head of s using $head(s)$, the next state of s is represented by $tail(s)$. We can ask ourselves if s' can match or imitate the action we just did for s, i.e. after observing the head of s', $head(s')$ is exactly $head(s)$, and we can repeat this process, but now starting at states $tail(s)$ and $tail(s')$. If we can indefinitely continue this process, then s' can *simulate* or *imitate* all observations encoded in s.

Now, to show that s can simulate all observations encoded in s', we reverse the roles of s' and s. In other words, we observe $head(s')$, which forces s' to change its state to $tail(s')$. Then, we ask ourselves if s can imitate the action we just did for s', or in other words, after observing the head of s, $head(s)$ is exactly $head(s')$, and we can repeat this process, but now starting at states $tail(s')$ and $tail(s)$.

This informal explanation is formally captured by the concepts of *bisimulation* and *bisimilar* streams, which we explain next.

Let $R \subseteq A^\omega \times A^\omega$ be a binary relation. We say that R is a *bisimulation* if the following two clauses hold:

- For every $(a, b) \in R$, we have $head(a) = head(b)$ and $(tail(a), tail(b)) \in R$.
- For every $(a, b) \in R^{-1}$, we have $head(a) = head(b)$ and $(tail(a), tail(b)) \in R^{-1}$.

where $R^{-1} = \{(b, a) \mid (a, b) \in R\}$ is the inverse of R. We say that s and s' in A^ω are *bisimilar* if there is some bisimulation $R \subseteq A^\omega \times A^\omega$ such that $(s, s') \in R$.

The first clause expresses that for any pair (a, b) in R, stream b simulates or imitates stream a, because b has the same head as a and we can repeat the process on their tails. The informal statement "repeat the process" is captured by condition $(tail(a), tail(b)) \in R$, because the first clause needs to be satisfied for *every* pair in R.

[43] The word *purely* is a convention in HoTT to emphasize that all our Σ-types are interpreted as constructive existentials, see Remark 1.5.30. We added the word "pure" to differentiate it from a (-1)-truncated version we will define later, i.e. a version that uses classical existentials, see Remark 1.11.5.

Similarly, the second clause states that for any pair (a, b) in R, stream a simulates or imitates stream b, because we are "reversing the roles" of the streams by working in the inverse relation R^{-1}.

In other words, R is a bisimulation if every pair in R consists of two elements in A^ω that "imitate or simulate each other". Hence, two streams are bisimilar if there is a bisimulation showing that they "simulate each other".

Notice that the second clause in the definition of bisimulation is superfluous, because given $(a, b) \in R^{-1}$ we can conclude $(b, a) \in R$, which implies that $head(b) = head(a)$ and $(tail(b), tail(a)) \in R$ by the first clause. Therefore, $(tail(a), tail(b)) \in R^{-1}$ as required. However, bear in mind that other coalgebras will have a non-superfluous second clause. It just happens that for the particular case of streams, their definition of bisimulation is quite simple.

Therefore, we can simplify the definition as follows. Let $R \subseteq A^\omega \times A^\omega$ be a binary relation. R is a *bisimulation* if for every $(a, b) \in R$, we have that $head(a) = head(b)$ and $(tail(a), tail(b)) \in R$.

In order to reach a definition of bisimulation that applies to arbitrary coalgebras, we now prove that stream bisimulations $R \subseteq A^\omega \times A^\omega$ are characterized by the following property. "R is a bisimulation *if and only if* R can be given a coalgebraic structure (i.e. there is a function $Out_R : R \to H_A(R)$) such that the first and second projections $pr_1, pr_2 : R \to A^\omega$ are coalgebra morphisms".

First, we prove direction $\Rightarrow$.

Define $Out_R(a, b) = (\ head(a),\ (tail(a),\ tail(b))\ )$. Notice that Out_R is well-defined, because whenever we have $(a, b) \in R$, it must hold $(tail(a),\ tail(b)) \in R$, since R is a bisimulation by hypothesis.

Now, pr_1 and pr_2 are coalgebra morphisms, because:

$$\begin{aligned} Out_{A^\omega}(pr_1(a, b)) &= (head(a), tail(a)) \\ &= H_A(pr_1)(\ head(a),\ (tail(a),\ tail(b))\ ) \\ &= H_A(pr_1)(Out_R(a, b)) \end{aligned}$$

$$\begin{aligned} Out_{A^\omega}(pr_2(a, b)) &= (head(b), tail(b)) \\ &= (head(a), tail(b)) \\ &= H_A(pr_2)(\ head(a),\ (tail(a),\ tail(b))\ ) \\ &= H_A(pr_2)(Out_R(a, b)) \end{aligned}$$

where we used the fact that $head(a) = head(b)$.

Now, we prove direction $\Leftarrow$.

Let $(a, b) \in R$. Since pr_1 and pr_2 are morphisms, we have:

$$\begin{aligned} H_A(pr_1)(Out_R(a, b)) &= Out_{A^\omega}(pr_1(a, b)) = (head(a), tail(a)) \\ H_A(pr_2)(Out_R(a, b)) &= Out_{A^\omega}(pr_2(a, b)) = (head(b), tail(b)) \end{aligned}$$

But we also have by definition of functor H_A:

$$\begin{aligned} H_A(pr_1)(Out_R(a, b)) &= H_A(pr_1)(\ pr_1(Out_R(a, b)),\ pr_2(Out_R(a, b))\ ) \\ &= (\ pr_1(Out_R(a, b)),\ pr_1(pr_2(Out_R(a, b)))\ ) \end{aligned}$$

$$\begin{aligned} H_A(pr_2)(Out_R(a, b)) &= H_A(pr_2)(\ pr_1(Out_R(a, b)),\ pr_2(Out_R(a, b))\ ) \\ &= (\ pr_1(Out_R(a, b)),\ pr_2(pr_2(Out_R(a, b)))\ ) \end{aligned}$$

Which implies:

$$\begin{aligned} head(a) = pr_1(Out_R(a, b)) &= head(b) \\ tail(a) &= pr_1(pr_2(Out_R(a, b))) \\ tail(b) &= pr_2(pr_2(Out_R(a, b))) \end{aligned}$$

But $(\ pr_1(pr_2(Out_R(a, b))),\ pr_2(pr_2(Out_R(a, b)))\ ) = pr_2(Out_R(a, b)) \in R$, which means $(tail(a), tail(b)) \in R$.

The generalization of bisimulation to arbitrary coalgebras is based on the above characterization for stream bisimulations. The following candidate definition is due to Mendler and Aczel (see Definition 2.7.1 in [17]).

Candidate definition 3.3.16 (Bisimulation, version 1). Let $H : \mathtt{SET} \to \mathtt{SET}$ be a functor. Let (A, Out_A) be an H-coalgebra.

Let $R \subseteq A \times A$ be a binary relation. We say that R is a bisimulation if there is a function $Out_R : R \to H(R)$ (hence, (R, Out_R) is an H-coalgebra) such that the first and second projections $pr_1, pr_2 : R \to A$ are coalgebra morphisms.

▲

In the literature, the standard concept of bisimulation is defined for relations $R \subseteq A \times B$, where A and B may be different sets. Instead, the above candidate definition makes use of relations in only one set $R \subseteq A \times A$. This simplified version will be enough for the purposes on this report.

Bisimulation can be rewritten in an equivalent form, as follows.

Candidate definition 3.3.17 (Bisimulation, version 2). Let $H : \mathtt{SET} \to \mathtt{SET}$ be a functor. Let (A, Out_A) be an H-coalgebra.

Let $R \subseteq A \times A$ be a binary relation. We say that R is a bisimulation if for every $(a, b) \in R$, there is $t \in H(R)$ such that $Out_A(a) = H(pr_1)(t)$ and $Out_A(b) = H(pr_2)(t)$.

▲

Classically, both versions of bisimulation are logically equivalent. For direction "version 1 $\Rightarrow$ version 2", define t to be $Out_R(a, b)$. For direction "version 1 $\Leftarrow$ version 2", use the axiom of choice to construct the function $Out_R : R \to H(R)$.

With the notion of bisimulation (version 2), we can now define bisimilarity.

Candidate definition 3.3.18 (Bisimilarity). Let $H : \mathtt{SET} \to \mathtt{SET}$ be a functor. Let (A, Out_A) be an H-coalgebra, and $x, y \in A$ two elements.

We say that x and y are bisimilar if there is a bisimulation $R \subseteq A \times A$ such that $(x, y) \in R$.

▲

Bisimulation (version 2) and bisimilarity can be expressed in HoTT as follows.

Definition 3.3.19 (Bisimulation)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k, m \leq i$ universe levels, $(\mathsf{A}, \mathsf{Out_A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ an H-coalgebra, and $\mathsf{R} : (\mathsf{A} \times \mathsf{A}) \to \mathcal{U}_m$ a binary type family.[44]

We define type:

$$\mathsf{IsBisimu\ R} :\equiv \prod_{\mathsf{a,b:A}} (\mathsf{R}\ (\mathsf{a}, \mathsf{b})) \to \left(\sum_{\mathsf{t:\ H} \sum_{\mathsf{z:A\times A}} \mathsf{R\ z}} (\mathsf{Out_A\ a} = \mathsf{map^H}\ (\mathsf{pr_1} \circ \mathsf{pr_1})\ \mathsf{t}) \times (\mathsf{Out_A\ b} = \mathsf{map^H}\ (\mathsf{pr_2} \circ \mathsf{pr_1})\ \mathsf{t}) \right)$$

When type $\mathsf{IsBisimu\ R}$ is inhabited, we say that R is a *bisimulation*.

▲

Type $\mathsf{H} \sum_{\mathsf{z:A\times A}} \mathsf{R\ z}$ represents $H(R)$ in Candidate definition 3.3.17. Notice $\sum_{\mathsf{z:A\times A}} \mathsf{R\ z}$ is at level $\max\{k, m\} \leq i$, which means H can be applied on it. Also, to extract the first and second coordinates of $\mathsf{A} \times \mathsf{A}$ in $\sum_{\mathsf{z:A\times A}} \mathsf{R\ z}$ we have to use $\mathsf{pr_1} \circ \mathsf{pr_1}$ and $\mathsf{pr_2} \circ \mathsf{pr_1}$, respectively.

Definition 3.3.20 (Bisimilarity)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k, m \leq i$ universe levels, and $(\mathsf{A}, \mathsf{Out_A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ an H-coalgebra.

Let $\mathsf{w} : \mathsf{A}$, and $\mathsf{y} : \mathsf{A}$ be two terms. We define type:

$$\mathsf{Bisim}_m\ \mathsf{w\ y} :\equiv \sum_{\mathsf{R:(A\times A)}\to\mathcal{U}_m} (\mathsf{R}\ (\mathsf{w}, \mathsf{y})) \times (\mathsf{IsBisimu\ R})$$

When type $(\mathsf{Bisim}_m\ \mathsf{w\ y})$ is inhabited, we say that w and y are *bisimilar*.

▲

[44] We will use type families of type $(\mathsf{A} \times \mathsf{A}) \to \mathbf{Prop}_m$ instead of $\mathsf{A} \to \mathsf{A} \to \mathbf{Prop}_m$, as this produces more symmetrical conditions in the definition of bisimulation.

Bisimilarity always implies pure behavioral equivalence, as the following lemma shows.

Lemma 3.3.21. *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor,* $k, m \leq i$ *universe levels,* $(\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ *an* H*-coalgebra, and* $\mathsf{w} : \mathsf{A}$, $\mathsf{y} : \mathsf{A}$ *two terms. Then, the following type is inhabited:*

$$(\mathsf{Bisim}_m\ \mathsf{w}\ \mathsf{y}) \to (\mathsf{PurelyBehEquiv}_{\max\{k,m\}}\ \mathsf{w}\ \mathsf{y})$$

Proof. By Σ-recursion, it is enough to prove $(\mathsf{PurelyBehEquiv}_{\max\{k,m\}}\ \mathsf{w}\ \mathsf{y})$ given $\mathsf{R} : (\mathsf{A} \times \mathsf{A}) \to \mathcal{U}_m$ with $\mathsf{h}_1 : \mathsf{R}\ (\mathsf{w}, \mathsf{y})$ and $\mathsf{h}_2 : \mathsf{IsBisimu}\ \mathsf{R}$.

First, we construct two surjective coalgebra morphisms:

$$\mathsf{m}_1, \mathsf{m}_2 : \mathsf{T} \to \mathsf{A}$$

where $(\mathsf{T}, \mathsf{Out}_\mathsf{T})$ is some H-coalgebra.

Define:

$$\mathsf{T} :\equiv \left(\sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{R}\ \mathsf{z}\right) + \left(\sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{pr}_1\ \mathsf{z} = \mathsf{pr}_2\ \mathsf{z}\right)$$

which is a type at level $\max\{k, m\} \leq i$ (see Remarks 1.5.29 and 1.5.37).

We need to define an Out function for T so that it becomes an H-coalgebra.

Define $\mathsf{f}_1 : \left(\sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{R}\ \mathsf{z}\right) \to (\mathsf{H}\ \mathsf{T})$ as the composition:

$$\left(\sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{R}\ \mathsf{z}\right) \xrightarrow{\mathsf{f}'_1} \left(\mathsf{H} \sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{R}\ \mathsf{z}\right) \xrightarrow{\mathsf{map}^\mathsf{H}\ \mathsf{inl}} \mathsf{H}\ \mathsf{T}$$

where inl is the left injection constructor for sum types, and f'_1 is defined by Σ-recursion (Lemma 1.5.25), applied twice:

$$\mathsf{f}'_1\ ((\mathsf{a}, \mathsf{b}), \mathsf{p}) :\equiv \mathsf{pr}_1\ (\mathsf{h}_2\ \mathsf{a}\ \mathsf{b}\ \mathsf{p})$$

for $\mathsf{a}, \mathsf{b} : \mathsf{A}$ and $\mathsf{p} : \mathsf{R}\ (\mathsf{a}, \mathsf{b})$.

Define $\mathsf{f}_2 : \left(\sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{pr}_1\ \mathsf{z} = \mathsf{pr}_2\ \mathsf{z}\right) \to (\mathsf{H}\ \mathsf{T})$ as the composition:

$$\left(\sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{pr}_1\ \mathsf{z} = \mathsf{pr}_2\ \mathsf{z}\right) \xrightarrow{\mathsf{pr}_1} \mathsf{A} \times \mathsf{A} \xrightarrow{\mathsf{pr}_1} \mathsf{A} \xrightarrow{\mathsf{Out}_\mathsf{A}} \mathsf{H}\ \mathsf{A} \xrightarrow{\mathsf{map}^\mathsf{H}\ \mathsf{f}'_2} \mathsf{H}\ \mathsf{T}$$

where f'_2 is defined as:

$$\mathsf{f}'_2 :\equiv (\lambda \mathsf{a} : \mathsf{A}.\ \mathsf{inr}\ ((\mathsf{a}, \mathsf{a}), \mathsf{refl}_\mathsf{A}\ \mathsf{a})) : \mathsf{A} \to \mathsf{T}$$

We can now define the $\mathsf{Out}_\mathsf{T} : \mathsf{T} \to (\mathsf{H}\ \mathsf{T})$ function by recursion on sum types (Lemma 1.5.36):

$$\mathsf{Out}_\mathsf{T}\ (\mathsf{inl}\ \mathsf{a}) :\equiv \mathsf{f}_1\ \mathsf{a}$$
$$\mathsf{Out}_\mathsf{T}\ (\mathsf{inr}\ \mathsf{b}) :\equiv \mathsf{f}_2\ \mathsf{b}$$

for $\mathsf{a} : \sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{R}\ \mathsf{z}$ and $\mathsf{b} : \sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{pr}_1\ \mathsf{z} = \mathsf{pr}_2\ \mathsf{z}$.

Now, we need to define our two surjective coalgebra morphisms $\mathsf{m}_1, \mathsf{m}_2 : \mathsf{T} \to \mathsf{A}$. We define them by recursion on sum types:

$$\mathsf{m}_1\ (\mathsf{inl}\ \mathsf{a}) :\equiv \mathsf{pr}_1\ (\mathsf{pr}_1\ \mathsf{a})$$
$$\mathsf{m}_1\ (\mathsf{inr}\ \mathsf{b}) :\equiv \mathsf{pr}_1\ (\mathsf{pr}_1\ \mathsf{b})$$

and:

$$\mathsf{m}_2\ (\mathsf{inl}\ \mathsf{a}) :\equiv \mathsf{pr}_2\ (\mathsf{pr}_1\ \mathsf{a})$$
$$\mathsf{m}_2\ (\mathsf{inr}\ \mathsf{b}) :\equiv \mathsf{pr}_2\ (\mathsf{pr}_1\ \mathsf{b})$$

Next, we prove that m_1 and m_2 are coalgebra morphisms (the case for m_2 is similar to the case for m_1, so we will only show the proof for m_1).

We have to prove that the following diagram commutes:

$$\begin{array}{ccc} \mathsf{T} & \xrightarrow{\mathsf{m}_1} & \mathsf{A} \\ {\scriptstyle \mathsf{Out}_\mathsf{T}}\downarrow & & \downarrow{\scriptstyle \mathsf{Out}_\mathsf{A}} \\ \mathsf{H}\,\mathsf{T} & \xrightarrow[\mathsf{map}^\mathsf{H}\ \mathsf{m}_1]{} & \mathsf{H}\,\mathsf{A} \end{array}$$

By induction on sum types (Lemma 1.5.35) we need to prove a left case and a right case.

- Given $\mathsf{q} : \sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{R}\ \mathsf{z}$, prove that the following type is inhabited:

$$\mathsf{Out}_\mathsf{A}\ (\mathsf{m}_1\ (\mathsf{inl}\ \mathsf{q})) = \mathsf{map}^\mathsf{H}\ \mathsf{m}_1\ (\mathsf{Out}_\mathsf{T}\ (\mathsf{inl}\ \mathsf{q}))$$

  By Σ-induction applied twice, it is enough to prove (for any $\mathsf{a}, \mathsf{b} : \mathsf{A}$ and $\mathsf{p} : \mathsf{R}\ (\mathsf{a}, \mathsf{b})$):

$$\mathsf{Out}_\mathsf{A}\ (\mathsf{m}_1\ (\mathsf{inl}\ ((\mathsf{a}, \mathsf{b}), \mathsf{p}))) = \mathsf{map}^\mathsf{H}\ \mathsf{m}_1\ (\mathsf{Out}_\mathsf{T}\ (\mathsf{inl}\ ((\mathsf{a}, \mathsf{b}), \mathsf{p})))$$

  But we have by definition of m_1:

$$\mathsf{Out}_\mathsf{A}\ (\mathsf{m}_1\ (\mathsf{inl}\ ((\mathsf{a}, \mathsf{b}), \mathsf{p}))) \equiv \mathsf{Out}_\mathsf{A}\ (\mathsf{pr}_1\ (\mathsf{pr}_1\ ((\mathsf{a}, \mathsf{b})\ \mathsf{p}))) \equiv \mathsf{Out}_\mathsf{A}\ \mathsf{a}$$

  And also:

$$\begin{aligned} \mathsf{map}^\mathsf{H}\ \mathsf{m}_1\ (\mathsf{Out}_\mathsf{T}\ (\mathsf{inl}\ ((\mathsf{a}, \mathsf{b}), \mathsf{p}))) &\overset{(1)}{=} \mathsf{map}^\mathsf{H}\ \mathsf{m}_1\ (\mathsf{f}_1\ ((\mathsf{a}, \mathsf{b}), \mathsf{p})) \\ &\overset{(2)}{=} \mathsf{map}^\mathsf{H}\ \mathsf{m}_1\ (\mathsf{map}^\mathsf{H}\ \mathsf{inl}\ (\mathsf{pr}_1\ (\mathsf{h}_2\ \mathsf{a}\ \mathsf{b}\ \mathsf{p}))) \\ &\overset{(3)}{=} \mathsf{map}^\mathsf{H}\ (\mathsf{m}_1 \circ \mathsf{inl})\ (\mathsf{pr}_1\ (\mathsf{h}_2\ \mathsf{a}\ \mathsf{b}\ \mathsf{p})) \\ &\overset{(4)}{=} \mathsf{map}^\mathsf{H}\ (\lambda \mathsf{t}.\ \mathsf{m}_1\ (\mathsf{inl}\ \mathsf{t}))\ (\mathsf{pr}_1\ (\mathsf{h}_2\ \mathsf{a}\ \mathsf{b}\ \mathsf{p})) \\ &\overset{(5)}{=} \mathsf{map}^\mathsf{H}\ (\lambda \mathsf{t}.\ \mathsf{pr}_1\ (\mathsf{pr}_1\ \mathsf{t}))\ (\mathsf{pr}_1\ (\mathsf{h}_2\ \mathsf{a}\ \mathsf{b}\ \mathsf{p})) \\ &\overset{(6)}{=} \mathsf{map}^\mathsf{H}\ (\mathsf{pr}_1 \circ \mathsf{pr}_1)\ (\mathsf{pr}_1\ (\mathsf{h}_2\ \mathsf{a}\ \mathsf{b}\ \mathsf{p})) \\ &\overset{(7)}{=} \mathsf{Out}_\mathsf{A}\ \mathsf{a} \end{aligned}$$

  where (1) follows by definition of Out_T, (2) by definition of f_1, (3) by functorial mapping of H, (4) by definition of function composition, (5) by definition of m_1, (6) by definition of function composition, and (7) holds because R is a bisimulation, i.e. we have a proof:

$$\mathsf{pr}_1\ (\mathsf{pr}_2\ (\mathsf{h}_2\ \mathsf{a}\ \mathsf{b}\ \mathsf{p})) : \mathsf{Out}_\mathsf{A}\ \mathsf{a} = \mathsf{map}^\mathsf{H}\ (\mathsf{pr}_1 \circ \mathsf{pr}_1)\ (\mathsf{pr}_1\ (\mathsf{h}_2\ \mathsf{a}\ \mathsf{b}\ \mathsf{p}))$$

- Given $\mathsf{q} : \sum_{\mathsf{z}:\mathsf{A}\times\mathsf{A}} \mathsf{pr}_1\ \mathsf{z} = \mathsf{pr}_2\ \mathsf{z}$, prove that the following type is inhabited:

$$\mathsf{Out}_\mathsf{A}\ (\mathsf{m}_1\ (\mathsf{inr}\ \mathsf{q})) = \mathsf{map}^\mathsf{H}\ \mathsf{m}_1\ (\mathsf{Out}_\mathsf{T}\ (\mathsf{inr}\ \mathsf{q}))$$

  By Σ-induction applied twice, it is enough to prove:

$$\mathsf{Out}_\mathsf{A}\ (\mathsf{m}_1\ (\mathsf{inr}\ ((\mathsf{a}, \mathsf{b}), \mathsf{p}))) = \mathsf{map}^\mathsf{H}\ \mathsf{m}_1\ (\mathsf{Out}_\mathsf{T}\ (\mathsf{inr}\ ((\mathsf{a}, \mathsf{b}), \mathsf{p})))$$

  for any $\mathsf{a}, \mathsf{b} : \mathsf{A}$ and $\mathsf{p} : \mathsf{pr}_1\ (\mathsf{a}, \mathsf{b}) = \mathsf{pr}_2\ (\mathsf{a}, \mathsf{b})$, or equivalently, $\mathsf{p} : \mathsf{a} = \mathsf{b}$.

  But we have by definition of m_1:

$$\mathsf{Out}_\mathsf{A}\ (\mathsf{m}_1\ (\mathsf{inr}\ ((\mathsf{a}, \mathsf{b}), \mathsf{p}))) \equiv \mathsf{Out}_\mathsf{A}\ (\mathsf{pr}_1\ (\mathsf{pr}_1\ ((\mathsf{a}, \mathsf{b})\ \mathsf{p}))) \equiv \mathsf{Out}_\mathsf{A}\ \mathsf{a}$$

And also:

$$\begin{aligned}
\mathsf{map}^{\mathsf{H}}\ \mathsf{m}_1\ (\mathsf{Out}_{\mathsf{T}}\ (\mathsf{inr}\ ((\mathsf{a},\mathsf{b}),\mathsf{p}))) &\overset{(1)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{m}_1\ (\mathsf{f}_2\ ((\mathsf{a},\mathsf{b}),\mathsf{p})) \\
&\overset{(2)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{m}_1\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{f}'_2\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a})) \\
&\overset{(3)}{=} \mathsf{map}^{\mathsf{H}}\ (\mathsf{m}_1 \circ \mathsf{f}'_2)\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a}) \\
&\overset{(4)}{=} \mathsf{map}^{\mathsf{H}}\ (\lambda \mathsf{t}.\ \mathsf{m}_1\ (\mathsf{f}'_2\ \mathsf{t}))\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a}) \\
&\overset{(5)}{=} \mathsf{map}^{\mathsf{H}}\ (\lambda \mathsf{t}.\ \mathsf{m}_1\ (\mathsf{inr}\ ((\mathsf{t},\mathsf{t}),\mathsf{refl}_{\mathsf{A}}\ \mathsf{t})))\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a}) \\
&\overset{(6)}{=} \mathsf{map}^{\mathsf{H}}\ (\lambda \mathsf{t}.\ \mathsf{t})\ (\mathsf{Out}_{\mathsf{A}}\ \mathsf{a}) \\
&\overset{(7)}{=} \mathsf{Out}_{\mathsf{A}}\ \mathsf{a}
\end{aligned}$$

where (1) follows by definition of $\mathsf{Out}_{\mathsf{T}}$, (2) by definition of f_2, (3) by functorial mapping of H, (4) by definition of function composition, (5) by definition of f'_2, (6) by definition of m_1, and (7) by functorial mapping of H on the identity function.

Now, we need to check that m_1 and m_2 are surjective functions (we will only show the proof for m_1, as the proof for m_2 is identical).

Let $\mathsf{a} : \mathsf{A}$, we have to show that the following type is inhabited:

$$\left\| \sum_{\mathsf{t}:\mathsf{T}} \mathsf{m}_1\ \mathsf{t} = \mathsf{a} \right\|$$

But we have $\mathsf{p}_1 :\equiv \mathsf{inr}\ ((\mathsf{a},\mathsf{a}),\mathsf{refl}_{\mathsf{A}}\ \mathsf{a}) : \mathsf{T}$, and also $\mathsf{refl}_{\mathsf{A}}\ \mathsf{a} : \mathsf{m}_1\ \mathsf{p}_1 = \mathsf{a}$, since $\mathsf{m}_1\ \mathsf{p}_1 \equiv \mathsf{m}_1\ (\mathsf{inr}\ ((\mathsf{a},\mathsf{a}),\mathsf{refl}_{\mathsf{A}}\ \mathsf{a})) \equiv \mathsf{a}$ by definition of m_1.

Hence, we have:

$$|(\mathsf{p}_1, \mathsf{refl}_{\mathsf{A}}\ \mathsf{a})| : \left\| \sum_{\mathsf{t}:\mathsf{T}} \mathsf{m}_1\ \mathsf{t} = \mathsf{a} \right\|$$

Therefore, we have a diagram of coalgebras and surjective coalgebra morphisms:

$$\begin{array}{ccc} \mathsf{T} & \xrightarrow{\mathsf{m}_2} & \mathsf{A} \\ {\scriptstyle \mathsf{m}_1}\downarrow & & \\ \mathsf{A} & & \end{array}$$

which, by Lemma 3.3.11, can be completed into a commutative "pushout square" of coalgebras and surjective coalgebra morphisms:

$$\begin{array}{ccc} \mathsf{T} & \xrightarrow{\mathsf{m}_2} & \mathsf{A} \\ {\scriptstyle \mathsf{m}_1}\downarrow & & \downarrow{\scriptstyle \mathsf{pr}} \\ \mathsf{A} & \xrightarrow[\mathsf{pl}]{} & \mathsf{Q} \end{array} \tag{3.25}$$

where Q is some coalgebra at level $\max\{k, m\}$.

Since we have two surjective coalgebra morphisms $\mathsf{pl}, \mathsf{pr} : \mathsf{A} \to \mathsf{Q}$, if we can prove that $\mathsf{pl}\ \mathsf{w} = \mathsf{pr}\ \mathsf{y}$ is inhabited, type $(\mathsf{PurelyBehEquiv}_{\max\{k,m\}}\ \mathsf{w}\ \mathsf{y})$ will be inhabited by definition.

But we have:

$$\begin{aligned}
\mathsf{pl}\ \mathsf{w} &\overset{(1)}{=} \mathsf{pl}\ (\mathsf{m}_1\ (\mathsf{inl}\ ((\mathsf{w},\mathsf{y}),\mathsf{h}_1))) \\
&\overset{(2)}{=} \mathsf{pr}\ (\mathsf{m}_2\ (\mathsf{inl}\ ((\mathsf{w},\mathsf{y}),\mathsf{h}_1))) \\
&\overset{(3)}{=} \mathsf{pr}\ \mathsf{y}
\end{aligned}$$

where (1) follows by definition of m_1,[45] (2) by commutativity of Diagram (3.25), and (3) by definition of m_2.

□

[45] In more detail, $\mathsf{m}_1\ (\mathsf{inl}\ ((\mathsf{w},\mathsf{y}),\mathsf{h}_1)) \equiv \mathsf{w}$, where $\mathsf{h}_1 : \mathsf{R}\ (\mathsf{w},\mathsf{y})$ is an hypothesis given at the start of the proof.

However, behavioral equivalence *does not* imply bisimilarity in general. For the concepts to coincide, functor H needs to preserve weak pullbacks (see Definition 175 and Theorem 177 in Chapter 6 of [7]).

Also, behavioral equivalence is always an equivalence relation.[46] However, on this report we will not need to make use of this fact.

To the contrary, bisimilarity is *not* an equivalence relation in general, as transitivity may fail. However, if functor H preserves weak pullbacks, bisimilarity will be an equivalence relation, since bisimilarity coincides with behavioral equivalence on this case (see Theorem 177 in Chapter 6 of [7]).

Behavioral equivalence has the following useful property.

Lemma 3.3.22. *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a functor,* $k, m \leq i$ *universe levels,* $(\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ *an* H*-coalgebra, and* $\mathsf{w} : \mathsf{A}$, $\mathsf{y} : \mathsf{A}$ *two terms.*

If $(\mathsf{PurelyBehEquiv}_m\ \mathsf{w}\ \mathsf{y})$ *is inhabited, then there is an* H*-coalgebra* $(\mathsf{Q}, \mathsf{Out}_\mathsf{Q}) : \mathsf{CoAlg}_{\max\{k,m\}}\ \mathsf{H}$ *and a surjective coalgebra morphism* $\mathsf{q} : \mathsf{A} \to \mathsf{Q}$ *such that* $\mathsf{q}\ \mathsf{w} = \mathsf{q}\ \mathsf{y}$.

Proof. By definition of $(\mathsf{PurelyBehEquiv}_m\ \mathsf{w}\ \mathsf{y})$, there is an H-coalgebra $(\mathsf{T}, \mathsf{Out}_\mathsf{T})$ at level m, and two surjective coalgebra morphisms:

$$\mathsf{A} \underset{\mathsf{g}}{\overset{\mathsf{h}}{\rightrightarrows}} \mathsf{T}$$

such that $\mathsf{h}\ \mathsf{w} = \mathsf{g}\ \mathsf{y}$ is inhabited.

By Lemma 3.3.12, this diagram can be completed into a commutative "coequalizer diagram" of coalgebras and surjective coalgebra morphisms:

$$\mathsf{A} \underset{\mathsf{g}}{\overset{\mathsf{h}}{\rightrightarrows}} \mathsf{T} \xrightarrow{\mathsf{p}} \mathsf{Q} \tag{3.26}$$

where $(\mathsf{Q}, \mathsf{Out}_\mathsf{Q})$ is some coalgebra at level $\max\{k, m\}$.

Now, define $\mathsf{q} :\equiv \mathsf{p} \circ \mathsf{h}$. It remains to show that q is a surjective coalgebra morphism and that $\mathsf{q}\ \mathsf{w} = \mathsf{q}\ \mathsf{y}$ is inhabited.

Function q is a coalgebra morphism because it is the composition of two coalgebra morphisms (Lemma 1.10.6).

Function q is surjective because it is the composition of two surjective functions (Lemma 1.11.11).

And we have:

$$\begin{aligned} \mathsf{q}\ \mathsf{w} &\overset{(1)}{=} \mathsf{p}\ (\mathsf{h}\ \mathsf{w}) \\ &\overset{(2)}{=} \mathsf{p}\ (\mathsf{g}\ \mathsf{y}) \\ &\overset{(3)}{=} \mathsf{p}\ (\mathsf{h}\ \mathsf{y}) \\ &\overset{(4)}{=} \mathsf{q}\ \mathsf{y} \end{aligned}$$

where (1) follows by definition of q, (2) holds since $\mathsf{h}\ \mathsf{w} = \mathsf{g}\ \mathsf{y}$ is inhabited by definition of $(\mathsf{PurelyBehEquiv}_m\ \mathsf{w}\ \mathsf{y})$, (3) follows by commutativity of Diagram (3.26), and (4) by definition of q.

□

One of our goals is to build a coinduction principle (this will be achieved at Step 3), which states something like: "If w and y behave the same way, then $\mathsf{w} = \mathsf{y}$". So, the natural way to proceed is to take a quotient over the "similar behavior" relation, as we want to identity terms that behave the same way. When A is a weakly final H-coalgebra (or LLWfinCoAlg), the quotient of A over the "similar behavior" relation will be our candidate for the final H-coalgebra (or LLFinCoAlg).

However, we have now two possible concepts for "similar behavior": behavioral equivalence and bisimilarity.[47] Which one should we use?

It is known that behavioral equivalence produces quotients that are coalgebras again [11]. Also, since behavioral equivalence is always an equivalence relation while bisimilarity may not be, it seems that behavioral equivalence is "better behaved" for the task. However, note that if we assume that functor H preserves weak pullbacks (see

[46] Reflexivity and symmetry follow directly from the definition. For a proof of transitivity, see Lemma 3.4 in [11].

[47] Remember that whenever a quotient is taken in HoTT (Lemma 1.11.24), the binary relation is not required to be an equivalence relation, as the quotient is defined over the smallest equivalence relation containing the given relation.

Therefore, we can still define a quotient using bisimilarity, even though bisimilarity fails to be an equivalence relation.

Definition 175 in Chapter 6 of [7]), we will be able to prove that H has a final coalgebra when bisimilarity is taken as the quotient relation. The reason is that bisimilarity and behavioral equivalence coincide when H preserves weak pullbacks. However, it is unknown to the author if we can get a final coalgebra by using bisimilarity when the weak pullback preservation property is dropped.

Therefore, we will use behavioral equivalence as our quotient relation. The reader might be asking what is the point of introducing bisimilarity if we are going to discard it anyway. Once we have a coinduction principle in terms of behavioral equivalence, then it is an easy corollary of Lemma 3.3.21 that we also have a coinduction principle that uses bisimilarity. It is easier to work with bisimilarity than with behavioral equivalence, as bisimilarity requires less conditions than behavioral equivalence.

To define a quotient over a relation, the relation is required to have type $\mathsf{A} \to \mathsf{A} \to \mathbf{Prop}_n$ (see Lemma 1.11.24), where n is some universe level. Therefore, given a functor $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ and an H-coalgebra $(\mathsf{A}, \mathsf{Out}_\mathsf{A})$ at level $k \leq i$, we need to (-1)-truncate pure behavioral equivalence:

$$\lambda(\mathsf{w} : \mathsf{A})(\mathsf{y} : \mathsf{A}).\ \|\mathsf{PurelyBehEquiv}_i\ \mathsf{w}\ \mathsf{y}\| \tag{3.27}$$

We know $\|\mathsf{PurelyBehEquiv}_i\ \mathsf{w}\ \mathsf{y}\|$ is a proposition at level $i+1$, because type $\mathsf{CoAlg}_i\ \mathsf{H}$ is at level $i+1$, type $\mathsf{CoAlgMor}\ \mathsf{A}\ \mathsf{T}$ is at level i for any coalgebra T at level i, and types $\mathsf{IsSurjection}\ \mathsf{f}$, $\mathsf{IsSurjection}\ \mathsf{g}$, and $\mathsf{f}\ \mathsf{w} = \mathsf{g}\ \mathsf{y}$ are at level i for any morphisms f and g.[48]

This means that the quotient of A over relation (3.27) will be at level $\max\{k, i+1\} = i+1 > i$ (see Lemma 1.11.24). But, for our development to work, we will require that the quotient does *not* exceed the input level i of functor H. Therefore, relation (3.27) will be *resized* as follows.

Definition 3.3.23 (Behavioral equivalence)**.** Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k \leq i$ a universe level, and $(\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ an H-coalgebra.

We define *behavioral equivalence* to be the binary relation:[49]

$$\mathsf{BehEquiv} :\equiv \lambda(\mathsf{w} : \mathsf{A})(\mathsf{y} : \mathsf{A}).\ \left(\mathsf{GPropR}_i^{i+1}\right)^{-1}\ \|\mathsf{PurelyBehEquiv}_i\ \mathsf{w}\ \mathsf{y}\|$$

having type $\mathsf{A} \to \mathsf{A} \to \mathbf{Prop}_i$. When proposition $(\mathsf{BehEquiv}\ \mathsf{w}\ \mathsf{y})$ is inhabited for $\mathsf{w}, \mathsf{y} : \mathsf{A}$, we say that w and y are *behaviorally equivalent*.

▲

We can now define the quotient.

Definition 3.3.24. Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k \leq i$ a universe level, and $(\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ an H-coalgebra. We define type:

$$\mathsf{A}_\mathsf{F} :\equiv (\mathsf{A}/\mathsf{BehEquiv}) : \mathcal{U}_i$$

▲

Term A_F is well-typed because it is at level $\max\{k, i\} = i$, by Lemma 1.11.24.

Step 2: Prove A_F is an H-coalgebra

First, we prove a lemma stating that the quotient constructor $[\,] : \mathsf{A} \to \mathsf{A}_\mathsf{F}$ factorizes through surjective coalgebra morphisms.

Lemma 3.3.25. *Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a functor, $k, m \leq i$ universe levels, and $(\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{CoAlg}_k\ \mathsf{H}$, $(\mathsf{B}, \mathsf{Out}_\mathsf{B}) : \mathsf{CoAlg}_m\ \mathsf{H}$ two H-coalgebras.*

Let $\mathsf{h} : \mathsf{A} \to \mathsf{B}$ be a surjective coalgebra morphism. Then, there is a function $\mathsf{g} : \mathsf{B} \to \mathsf{A}_\mathsf{F}$ such that $[\,] \sim (\mathsf{g} \circ \mathsf{h})$.

Proof. Since there is no obvious recursion principle to help us define a function out of B (as B is arbitrary), we will use the Principle of unique choice (Lemma 1.11.8) and the Choice Theorem (Theorem 1.11.6) to define g.

Define:

$$\mathsf{P} :\equiv \lambda \mathsf{w} : \mathsf{B}.\ \sum_{\mathsf{y}:\mathsf{A}_\mathsf{F}} \left\| \sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}] = \mathsf{y}) \times (\mathsf{h}\ \mathsf{c} = \mathsf{w}) \right\|$$

[48] Regarding computation of universe levels, see Remarks 1.5.16, 1.5.29, Definition 1.11.1, and the formation rule for identity types on page 45.

[49] GPropR_i^{i+1} is the propositional resizing equivalence of Lemma 1.8.22.

P is stating what g should do to every input w: "The required function should map $\mathsf{w}\colon \mathsf{B}$ into some $\mathsf{y}\colon \mathsf{A_F}$ such that y can be put in the form $[\mathsf{c}]$ for some $\mathsf{c}\colon \mathsf{A}$, in such a way that $\mathsf{h\ c} = \mathsf{w}$, i.e. if we had our function g, then $\mathsf{g\ w} = \mathsf{g\ (h\ c)} = [\mathsf{c}]$".

Claim 1: Type $\prod_{\mathsf{w}:\mathsf{B}} \mathsf{IsProp}\ (\mathsf{P\ w})$ is inhabited.
Let $\mathsf{w}\colon \mathsf{B}$. We need to prove $\prod_{\mathsf{p_1},\mathsf{p_2}:\mathsf{P\ w}} (\mathsf{p_1} = \mathsf{p_2})$. By Σ-induction (Lemma 1.5.24), it is enough to prove:

$$\prod_{\mathsf{y_1}:\mathsf{A_F}} \prod_{\mathsf{py_1}:\left\|\sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}]=\mathsf{y_1})\times(\mathsf{h\ c}=\mathsf{w})\right\|} \prod_{\mathsf{y_2}:\mathsf{A_F}} \prod_{\mathsf{py_2}:\left\|\sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}]=\mathsf{y_2})\times(\mathsf{h\ c}=\mathsf{w})\right\|} (\mathsf{y_1},\mathsf{py_1}) = (\mathsf{y_2},\mathsf{py_2})$$

Since $\left\|\sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}] = \mathsf{y}) \times (\mathsf{h\ c} = \mathsf{w})\right\|$ is a mere proposition for every $\mathsf{y}\colon \mathsf{A_F}$, by Lemma 1.8.14, it is enough to prove:

$$\prod_{\mathsf{y_1}:\mathsf{A_F}} \prod_{\mathsf{py_1}:\left\|\sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}]=\mathsf{y_1})\times(\mathsf{h\ c}=\mathsf{w})\right\|} \prod_{\mathsf{y_2}:\mathsf{A_F}} \prod_{\mathsf{py_2}:\left\|\sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}]=\mathsf{y_2})\times(\mathsf{h\ c}=\mathsf{w})\right\|} \mathsf{y_1} = \mathsf{y_2}$$

or equivalently, after simplifying notation and rearranging quantified variables:

$$\prod_{\mathsf{y_1},\mathsf{y_2}:\mathsf{A_F}} \left\|\sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}] = \mathsf{y_1}) \times (\mathsf{h\ c} = \mathsf{w})\right\| \to \left\|\sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}] = \mathsf{y_2}) \times (\mathsf{h\ c} = \mathsf{w})\right\| \to (\mathsf{y_1} = \mathsf{y_2})$$

Let $\mathsf{y_1}, \mathsf{y_2}\colon \mathsf{A_F}$. We know $(\mathsf{y_1} = \mathsf{y_2})$ is a mere proposition, since $\mathsf{A_F}$ is a set (see Lemmas 1.11.24 and 1.8.17). Also, since mere propositions are closed under the arrow type (Lemma 1.8.17), by (-1)-recursion (Corollary 1.11.4) applied twice, it is enough to prove:

$$\left(\sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}] = \mathsf{y_1}) \times (\mathsf{h\ c} = \mathsf{w})\right) \to \left(\sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}] = \mathsf{y_2}) \times (\mathsf{h\ c} = \mathsf{w})\right) \to (\mathsf{y_1} = \mathsf{y_2})$$

but, by Σ-recursion (Lemma 1.5.25), it is enough to prove:

$$\prod_{\mathsf{c_1},\mathsf{c_2}:\mathsf{A}} ([\mathsf{c_1}] = \mathsf{y_1}) \to (\mathsf{h\ c_1} = \mathsf{w}) \to ([\mathsf{c_2}] = \mathsf{y_2}) \to (\mathsf{h\ c_2} = \mathsf{w}) \to (\mathsf{y_1} = \mathsf{y_2})$$

Let $\mathsf{c_1}, \mathsf{c_2}$:A with the following inhabited types:

$$\begin{aligned} [\mathsf{c_1}] &= \mathsf{y_1} \qquad \mathsf{h\ c_1} = \mathsf{w} \\ [\mathsf{c_2}] &= \mathsf{y_2} \qquad \mathsf{h\ c_2} = \mathsf{w} \end{aligned} \tag{3.28}$$

Therefore, to prove $(\mathsf{y_1} = \mathsf{y_2})$, it is enough to prove $[\mathsf{c_1}] = [\mathsf{c_2}]$. But, by the higher constructor for quotients $\mathsf{quotiden}$ (see Lemma 1.11.24), it is enough to prove that $(\mathsf{BehEquiv\ c_1\ c_2})$ is inhabited.

By Lemma 1.8.23 and the (-1)-truncation constructor $\|\ \|$ (Definition 1.11.1), it is enough to prove that $(\mathsf{PurelyBehEquiv}_i\ \mathsf{c_1\ c_2})$ is inhabited.

By hypothesis, we have a coalgebra $(\mathsf{B}, \mathsf{Out_B})$ at level $m \leq i$ (hence, at level i), and a surjective coalgebra morphism $\mathsf{h}\colon \mathsf{A} \to \mathsf{B}$. Therefore, by definition of $\mathsf{PurelyBehEquiv}_i\ \mathsf{c_1\ c_2}$, it remains to show that type $\mathsf{h\ c_1} = \mathsf{h\ c_2}$ is inhabited (we use h twice in the definition of $\mathsf{PurelyBehEquiv}_i\ \mathsf{c_1\ c_2}$).

But $\mathsf{h\ c_1} = \mathsf{w}$ and $\mathsf{w} = \mathsf{h\ c_2}$ are inhabited by (3.28).

This proves the claim.

Claim 2: Type $\prod_{\mathsf{w}:\mathsf{B}} \|\mathsf{P\ w}\|$ is inhabited.
Let $\mathsf{w}\colon \mathsf{B}$. Since h is surjective, type $\left\|\sum_{\mathsf{c}:\mathsf{A}} \mathsf{h\ c} = \mathsf{w}\right\|$ is inhabited. So, we will be done if we can prove that the following type is inhabited:

$$\left\|\sum_{\mathsf{c}:\mathsf{A}} \mathsf{h\ c} = \mathsf{w}\right\| \to \|\mathsf{P\ w}\|$$

By (-1)-truncation recursion (Corollary 1.11.4) and Σ-recursion, it is enough to prove:

$$\prod_{c:A} (\mathsf{h}\ c = w) \to \|P\ w\|$$

Let $c : A$ with $cp : \mathsf{h}\ c = w$. Then, we have:

$$|(\ [c],\ |(c, \mathsf{refl}_{A_F}\ [c], cp)|\ )| : \|P\ w\|$$

This proves the claim.

Therefore, by Claims 1 and 2 and the Principle of unique choice (Lemma 1.11.8), the following type is inhabited:

$$\prod_{w:B} \sum_{y:A_F} \left\| \sum_{c:A} ([c] = y) \times (\mathsf{h}\ c = w) \right\|$$

Then, by Theorem 1.11.6, the following type is inhabited:

$$\sum_{g:B\to A_F} \prod_{w:B} \left\| \sum_{c:A} ([c] = g\ w) \times (\mathsf{h}\ c = w) \right\|$$

In other words, we have a function $g : B \to A_F$, and a term:

$$t : \prod_{w:B} \left\| \sum_{c:A} ([c] = g\ w) \times (\mathsf{h}\ c = w) \right\|$$

It remains to show that $[\,] \sim (g \circ \mathsf{h})$ is inhabited. Let $z : A$. We have to prove $[z] = g\ (\mathsf{h}\ z)$. First, we have:

$$t\ (\mathsf{h}\ z) : \left\| \sum_{c:A} ([c] = g\ (\mathsf{h}\ z)) \times (\mathsf{h}\ c = \mathsf{h}\ z) \right\|$$

Therefore, we will be done if we can prove that the following type is inhabited:

$$\left\| \sum_{c:A} ([c] = g\ (\mathsf{h}\ z)) \times (\mathsf{h}\ c = \mathsf{h}\ z) \right\| \to ([z] = g\ (\mathsf{h}\ z))$$

We know $[z] = g\ (\mathsf{h}\ z)$ is a mere proposition, because A_F is a set. Hence, by (-1)-truncation recursion and Σ-recursion, it is enough to prove:

$$\prod_{c:A} ([c] = g\ (\mathsf{h}\ z)) \to (\mathsf{h}\ c = \mathsf{h}\ z) \to ([z] = g\ (\mathsf{h}\ z))$$

Let $c : A$ with the following inhabited types:

$$[c] = g\ (\mathsf{h}\ z) \qquad \mathsf{h}\ c = \mathsf{h}\ z \tag{3.29}$$

By hypothesis, we have a coalgebra (B, Out_B) at level $m \leq i$ (hence, at level i), and a surjective coalgebra morphism $\mathsf{h} : A \to B$. Also, we have $\mathsf{h}\ c = \mathsf{h}\ z$ by (3.29), which means $(\mathsf{PurelyBehEquiv}_i\ c\ z)$ is inhabited (we use h twice in the definition of $\mathsf{PurelyBehEquiv}_i\ c\ z$).

Hence, by the (-1)-truncation constructor $|\ |$ and Lemma 1.8.23, type $(\mathsf{BehEquiv}\ c\ z)$ is inhabited. Then, by the higher constructor for quotients $\mathsf{quotiden}$, type $[c] = [z]$ is inhabited.

So, we have $[z] = [c]$ and $[c] = g\ (\mathsf{h}\ z)$ by (3.29). Therefore, $[z] = g\ (\mathsf{h}\ z)$ is inhabited.

□

We proceed to prove that A_F is an H-coalgebra.

Lemma 3.3.26. *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a set-preserving*[50] *functor,* $k \leq i$ *a universe level, and* $(\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ *an* H*-coalgebra. Then, there is a function* $\mathsf{Out}_{\mathsf{A}_\mathsf{F}} : \mathsf{A}_\mathsf{F} \to (\mathsf{H}\ \mathsf{A}_\mathsf{F})$.

In other words, $(\mathsf{A}_\mathsf{F}, \mathsf{Out}_{\mathsf{A}_\mathsf{F}})$ *is an* H*-coalgebra at level* i.

Proof. We will define $\mathsf{Out}_{\mathsf{A}_\mathsf{F}}$ by the recursion principle for quotients (Lemma 1.11.26).

Observe that $\mathsf{H}\ \mathsf{A}_\mathsf{F}$ is a set, because H is a set-preserving functor and A_F is a set (see Lemma 1.11.24).

First, define function $\mathsf{h} : \mathsf{A} \to (\mathsf{H}\ \mathsf{A}_\mathsf{F})$ as the composition:

$$\mathsf{A} \xrightarrow{\mathsf{Out}_\mathsf{A}} \mathsf{H}\ \mathsf{A} \xrightarrow{\mathsf{map}^\mathsf{H}\ [\,]} \mathsf{H}\ \mathsf{A}_\mathsf{F}$$

If we can prove that the following type is inhabited:

$$\prod_{\mathsf{a},\mathsf{b}:\mathsf{A}} (\mathsf{BehEquiv}\ \mathsf{a}\ \mathsf{b}) \to (\mathsf{h}\ \mathsf{a} = \mathsf{h}\ \mathsf{b})$$

or equivalently:

$$\prod_{\mathsf{a},\mathsf{b}:\mathsf{A}} (\mathsf{BehEquiv}\ \mathsf{a}\ \mathsf{b}) \to (\mathsf{map}^\mathsf{H}\ [\,]\ (\mathsf{Out}_\mathsf{A}\ \mathsf{a}) = \mathsf{map}^\mathsf{H}\ [\,]\ (\mathsf{Out}_\mathsf{A}\ \mathsf{b}))$$

then, by the recursion principle for quotients, there will be a function $\mathsf{Out}_{\mathsf{A}_\mathsf{F}} : \mathsf{A}_\mathsf{F} \to (\mathsf{H}\ \mathsf{A}_\mathsf{F})$, such that for every $\mathsf{a} : \mathsf{A}$ it will satisfy:

$$\mathsf{Out}_{\mathsf{A}_\mathsf{F}}\ [\mathsf{a}] \equiv \mathsf{h}\ \mathsf{a} \equiv \mathsf{map}^\mathsf{H}\ [\,]\ (\mathsf{Out}_\mathsf{A}\ \mathsf{a}) \tag{3.30}$$

Let $\mathsf{a}, \mathsf{b} : \mathsf{A}$. By Lemma 1.8.24 and (-1)-truncation recursion (Corollary 1.11.4), it is enough to prove:

$$(\mathsf{PurelyBehEquiv}_i\ \mathsf{a}\ \mathsf{b}) \to (\mathsf{map}^\mathsf{H}\ [\,]\ (\mathsf{Out}_\mathsf{A}\ \mathsf{a}) = \mathsf{map}^\mathsf{H}\ [\,]\ (\mathsf{Out}_\mathsf{A}\ \mathsf{b}))$$

So, suppose $(\mathsf{PurelyBehEquiv}_i\ \mathsf{a}\ \mathsf{b})$. Then, by Lemma 3.3.22, there is an H-coalgebra $(\mathsf{Q}, \mathsf{Out}_\mathsf{Q})$ at level $\max\{k, i\} = i$ and a surjective coalgebra morphism $\mathsf{q} : \mathsf{A} \to \mathsf{Q}$ such that $\mathsf{q}\ \mathsf{a} = \mathsf{q}\ \mathsf{b}$ is inhabited.

But then, by Lemma 3.3.25, the quotient constructor $[\,]$ factorizes through q, i.e. there is some function $\mathsf{g} : \mathsf{Q} \to \mathsf{A}_\mathsf{F}$ such that $[\,] \sim (\mathsf{g} \circ \mathsf{q})$. Hence, $[\,] = \mathsf{g} \circ \mathsf{q}$ is inhabited by function extensionality.

Therefore, we have:

$$\begin{aligned}
\mathsf{map}^\mathsf{H}\ [\,]\ (\mathsf{Out}_\mathsf{A}\ \mathsf{a}) &\overset{(1)}{=} \mathsf{map}^\mathsf{H}\ (\mathsf{g} \circ \mathsf{q})\ (\mathsf{Out}_\mathsf{A}\ \mathsf{a}) \\
&\overset{(2)}{=} \mathsf{map}^\mathsf{H}\ \mathsf{g}\ (\mathsf{map}^\mathsf{H}\ \mathsf{q}\ (\mathsf{Out}_\mathsf{A}\ \mathsf{a})) \\
&\overset{(3)}{=} \mathsf{map}^\mathsf{H}\ \mathsf{g}\ (\mathsf{Out}_\mathsf{Q}\ (\mathsf{q}\ \mathsf{a})) \\
&\overset{(4)}{=} \mathsf{map}^\mathsf{H}\ \mathsf{g}\ (\mathsf{Out}_\mathsf{Q}\ (\mathsf{q}\ \mathsf{b})) \\
&\overset{(5)}{=} \mathsf{map}^\mathsf{H}\ \mathsf{g}\ (\mathsf{map}^\mathsf{H}\ \mathsf{q}\ (\mathsf{Out}_\mathsf{A}\ \mathsf{b})) \\
&\overset{(6)}{=} \mathsf{map}^\mathsf{H}\ (\mathsf{g} \circ \mathsf{q})\ (\mathsf{Out}_\mathsf{A}\ \mathsf{b}) \\
&\overset{(7)}{=} \mathsf{map}^\mathsf{H}\ [\,]\ (\mathsf{Out}_\mathsf{A}\ \mathsf{b})
\end{aligned}$$

where (1) and (7) hold because $[\,] = \mathsf{g} \circ \mathsf{q}$ is inhabited, (2) and (6) hold by functorial mapping of H, (3) and (5) hold because $\mathsf{q} : \mathsf{A} \to \mathsf{Q}$ is a coalgebra morphism, and (4) holds because $\mathsf{q}\ \mathsf{a} = \mathsf{q}\ \mathsf{b}$ is inhabited.

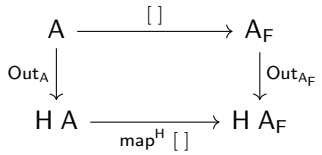
□

We have the following corollary.

Corollary 3.3.27. *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a set-preserving functor,* $k \leq i$ *a universe level, and* $(\mathsf{A}, \mathsf{Out}_\mathsf{A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ *an* H*-coalgebra. Then, the quotient constructor* $[\,] : \mathsf{A} \to \mathsf{A}_\mathsf{F}$ *is a coalgebra morphism.*

[50] See Definition 1.9.9.

In other words, we have a commutative diagram:



Proof. By definition of $\mathsf{Out}_{\mathsf{A_F}}$, as it satisfies equation (3.30) in the proof of Lemma 3.3.26.

□

Step 3: Construct a coinduction principle for $\mathsf{A_F}$

We will have two coinduction principles, one using behavioral equivalence, and another one using bisimilarity.

Lemma 3.3.28 (Coinduction principle, behavioral equivalence version)**.** *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a set-preserving functor,* $k, m \leq i$ *universe levels, and* $(\mathsf{A}, \mathsf{Out_A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ *an* H*-coalgebra. Then, the following type is inhabited:*[51]

$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{A_F}} (\mathsf{PurelyBehEquiv}_m\ \mathsf{w}\ \mathsf{y}) \to (\mathsf{w} = \mathsf{y})$$

Proof. Since $\mathsf{A_F}$ is a set, type $\mathsf{w} = \mathsf{y}$ is a mere proposition for every $\mathsf{w}, \mathsf{y} : \mathsf{A_F}$. Also, mere propositions are closed under Π-types by Lemma 1.8.17. Therefore, we can apply the induction principle for quotients (Lemma 1.11.25) twice, so that it is enough to prove:

$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{A}} (\mathsf{PurelyBehEquiv}_m\ [\mathsf{w}]\ [\mathsf{y}]) \to ([\mathsf{w}] = [\mathsf{y}])$$

Let $\mathsf{w}, \mathsf{y} : \mathsf{A}$ with $(\mathsf{PurelyBehEquiv}_m\ [\mathsf{w}]\ [\mathsf{y}])$. To prove $[\mathsf{w}] = [\mathsf{y}]$, it is enough to prove $(\mathsf{BehEquiv}\ \mathsf{w}\ \mathsf{y})$ by the higher constructor quotiden for quotients (see Lemma 1.11.24).

By Lemma 1.8.23 and the (-1)-truncation constructor $|\ |$, it is enough to prove $(\mathsf{PurelyBehEquiv}_i\ \mathsf{w}\ \mathsf{y})$.

Now, since $(\mathsf{PurelyBehEquiv}_m\ [\mathsf{w}]\ [\mathsf{y}])$ is inhabited, there is an H-coalgebra $(\mathsf{T}, \mathsf{Out_T})$ at level m (hence, at level i) and two surjective coalgebra morphisms $\mathsf{h}, \mathsf{g} : \mathsf{A_F} \to \mathsf{T}$ such that $\mathsf{h}\ [\mathsf{w}] = \mathsf{g}\ [\mathsf{y}]$ is inhabited.

But then, we have the compositions $\mathsf{h} \circ [\ ] : \mathsf{A} \to \mathsf{T}$ and $\mathsf{g} \circ [\ ] : \mathsf{A} \to \mathsf{T}$. Functions $\mathsf{h} \circ [\ ]$ and $\mathsf{g} \circ [\ ]$ are surjective coalgebra morphisms, since $[\ ]$ is a coalgebra morphism (Corollary 3.3.27), coalgebra morphisms compose (Lemma 1.10.6), $[\ ]$ is a surjective function (Lemma 1.11.27), and surjective functions compose (Lemma 1.11.11).

Therefore, to conclude that $(\mathsf{PurelyBehEquiv}_i\ \mathsf{w}\ \mathsf{y})$ is inhabited, it remains to show that type $(\mathsf{h} \circ [\ ])\ \mathsf{w} = (\mathsf{g} \circ [\ ])\ \mathsf{y}$ is inhabited. But, by definition of composition, this simplifies to $\mathsf{h}\ [\mathsf{w}] = \mathsf{g}\ [\mathsf{y}]$, which is already inhabited.

□

As a corollary of Lemma 3.3.21, we have the bisimilarity version.

Corollary 3.3.29 (Coinduction principle, bisimilarity version)**.** *Let* $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ *be a set-preserving functor,* $k, m \leq i$ *universe levels, and* $(\mathsf{A}, \mathsf{Out_A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ *an* H*-coalgebra. Then, the following type is inhabited:*

$$\prod_{\mathsf{w},\mathsf{y}:\mathsf{A_F}} (\mathsf{Bisim}_m\ \mathsf{w}\ \mathsf{y}) \to (\mathsf{w} = \mathsf{y})$$

Proof. Let $\mathsf{w}, \mathsf{y} : \mathsf{A_F}$ with $(\mathsf{Bisim}_m\ \mathsf{w}\ \mathsf{y})$. By Lemma 3.3.21 applied with $(\mathsf{A_F}, \mathsf{Out}_{\mathsf{A_F}}) : \mathsf{CoAlg}_i\ \mathsf{H}$, we have that $(\mathsf{PurelyBehEquiv}_{\max\{i,m\}}\ \mathsf{w}\ \mathsf{y})$, or equivalently, $(\mathsf{PurelyBehEquiv}_i\ \mathsf{w}\ \mathsf{y})$ is inhabited.

Therefore, by Lemma 3.3.28 applied with $(\mathsf{A}, \mathsf{Out_A}) : \mathsf{CoAlg}_k\ \mathsf{H}$ and $m = i$, we have that $(\mathsf{w} = \mathsf{y})$ is inhabited.

□

[51] Notice that $(\mathsf{PurelyBehEquiv}_m\ \mathsf{w}\ \mathsf{y})$ is applied on $\mathsf{A_F}$, *not* on A. This is valid since $(\mathsf{A_F}, \mathsf{Out}_{\mathsf{A_F}})$ is an H-coalgebra at level i by Lemma 3.3.26.

Step 4: Prove finality for $\mathsf{A_F}$

All previous steps worked with an arbitrary H-coalgebra. For this step we require the stronger condition that the coalgebra is a LLWfinCoAlg or a WfinCoAlg. This means we will prove two cases, i.e. if A is a LLWfinCoAlg, then $\mathsf{A_F}$ is a LLFinCoAlg; if A is a WfinCoAlg, then $\mathsf{A_F}$ is a FinCoAlg.

Theorem 3.3.30. *Let $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ be a set-preserving functor.*

(i) Let $i > 0$ and $(\mathsf{A}, \mathsf{Out_A}, \mathsf{m}) : \mathsf{LLWfinCoAlg}_i\ \mathsf{H}$ a lower level weakly final H-coalgebra. Then the tuple $(\mathsf{A_F}, \mathsf{Out_{A_F}}, \mathsf{mc})$ is a lower level final H-coalgebra, where mc is a proof that the type of morphisms going into $\mathsf{A_F}$ is contractible.

In other words, we have the tuple:[52]

$$(\mathsf{A_F}, \mathsf{Out_{A_F}}, \mathsf{mc}) : \mathsf{LLFinCoAlg}_i\ \mathsf{H}$$

(ii) Let $(\mathsf{A}, \mathsf{Out_A}, \mathsf{m}) : \mathsf{WfinCoAlg}_i\ \mathsf{H}$ be a weakly final H-coalgebra. Then the tuple $(\mathsf{A_F}, \mathsf{Out_{A_F}}, \mathsf{mc})$ is a final H-coalgebra, where mc is a proof that the type of morphisms going into $\mathsf{A_F}$ is contractible.

In other words, we have the tuple:

$$(\mathsf{A_F}, \mathsf{Out_{A_F}}, \mathsf{mc}) : \mathsf{FinCoAlg}_i\ \mathsf{H}$$

Proof. The proof for both results is almost identical. As such, we will develop the proof for (i). The proof for (ii) is obtained by replacing all occurrences of universe level $i-1$ for universe level i.

We have to construct a proof:

$$\mathsf{mc} : \prod_{(\mathsf{B},\mathsf{Out_B}):\mathsf{CoAlg}_{i-1}\ \mathsf{H}} \mathsf{IsContr}\ (\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{A_F})$$

Let $(\mathsf{B}, \mathsf{Out_B}) : \mathsf{CoAlg}_{i-1}\ \mathsf{H}$.

We have to build a term of type $\mathsf{IsContr}\ (\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{A_F})$. By definition of contractible type, this means we have to build a term of type:

$$\sum_{(\mathsf{h},\mathsf{w}_1):\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{A_F}} \prod_{(\mathsf{g},\mathsf{w}_2):\mathsf{CoAlgMor}\ \mathsf{B}\ \mathsf{A_F}} (\mathsf{h}, \mathsf{w}_1) = (\mathsf{g}, \mathsf{w}_2)$$

Intuitively, this means we have to prove the "existence" of a coalgebra morphism $(\mathsf{h}, \mathsf{w}_1)$ from B to $\mathsf{A_F}$, and the "uniqueness" of such morphism.

Existence

By hypothesis, we have a term:

$$\mathsf{m} : \prod_{(\mathsf{C},\mathsf{Out_C}):\mathsf{CoAlg}_{i-1}\ \mathsf{H}} \mathsf{CoAlgMor}\ \mathsf{C}\ \mathsf{A}$$

This means we have a function:

$$\mathsf{pr}_1\ (\mathsf{m}\ (\mathsf{B}, \mathsf{Out_B})) : \mathsf{B} \to \mathsf{A}$$

and also $\mathsf{pr}_2\ (\mathsf{m}\ (\mathsf{B}, \mathsf{Out_B}))$ is a proof that this function is a morphism.

Therefore, we can define h as the composition:

$$\mathsf{B} \xrightarrow{\mathsf{pr}_1\ (\mathsf{m}\ (\mathsf{B},\mathsf{Out_B}))} \mathsf{A} \xrightarrow{[\,]} \mathsf{A_F}$$

It remains to build a proof w_1 that h is a morphism. But h is the composition of two morphisms, since $[\,]$ is a coalgebra morphism by Corollary 3.3.27, and $\mathsf{pr}_2\ (\mathsf{m}\ (\mathsf{B}, \mathsf{Out_B}))$ is a proof that $\mathsf{pr}_1\ (\mathsf{m}\ (\mathsf{B}, \mathsf{Out_B}))$ is a morphism.

[52] By Lemma 3.3.26, we have that $(\mathsf{A_F}, \mathsf{Out_{A_F}})$ is a coalgebra at universe level i.

Therefore, by Lemma 1.10.6, h is a coalgebra morphism.[53] This means we have the commutative diagram:

$$\begin{array}{ccc} \mathsf{B} & \xrightarrow{\mathsf{h}} & \mathsf{A_F} \\ {\scriptstyle \mathsf{Out_B}}\downarrow & & \downarrow{\scriptstyle \mathsf{Out_{A_F}}} \\ \mathsf{H\ B} & \xrightarrow[\mathsf{map^H\ h}]{} & \mathsf{H\ A_F} \end{array} \tag{3.31}$$

Uniqueness

We have to show that the following type is inhabited:

$$\prod_{(\mathsf{g},\mathsf{w}_2):\mathsf{CoAlgMor\ B\ A_F}} (\mathsf{h},\mathsf{w}_1) = (\mathsf{g},\mathsf{w}_2)$$

where $(\mathsf{h},\mathsf{w}_1)$ is the morphism built in the existence part.

Let $(\mathsf{g},\mathsf{w}_2) : \mathsf{CoAlgMor\ B\ A_F}$. In other words, we have the commutative diagram:

$$\begin{array}{ccc} \mathsf{B} & \xrightarrow{\mathsf{g}} & \mathsf{A_F} \\ {\scriptstyle \mathsf{Out_B}}\downarrow & & \downarrow{\scriptstyle \mathsf{Out_{A_F}}} \\ \mathsf{H\ B} & \xrightarrow[\mathsf{map^H\ g}]{} & \mathsf{H\ A_F} \end{array} \tag{3.32}$$

We know that type $\mathsf{CoAlgMor\ B\ A_F}$ is defined as (see Definition 1.10.5):

$$\sum_{\mathsf{f}:\mathsf{B}\to\mathsf{A_F}} \prod_{\mathsf{w}:\mathsf{B}} \mathsf{Out_{A_F}}\ (\mathsf{f\ w}) = \mathsf{map^H\ f}\ (\mathsf{Out_B\ w})$$

The second component of this Σ-type, i.e.,

$$\prod_{\mathsf{w}:\mathsf{B}} \mathsf{Out_{A_F}}\ (\mathsf{f\ w}) = \mathsf{map^H\ f}\ (\mathsf{Out_B\ w})$$

is a mere proposition for every $\mathsf{f} : \mathsf{B} \to \mathsf{A_F}$, since both terms appearing in the identity are of type $\mathsf{H\ A_F}$, which is a set ($\mathsf{A_F}$ is a set and H is set-preserving), and propositions are closed under Π-types (see Lemma 1.8.17).

Therefore, by Lemma 1.8.14, to prove that $(\mathsf{h},\mathsf{w}_1) = (\mathsf{g},\mathsf{w}_2)$ is inhabited, it is enough to prove that $\mathsf{h} = \mathsf{g}$ holds. However, by function extensionality (see Section 1.6.1), it is enough to prove:

$$\prod_{\mathsf{w}:\mathsf{B}} \mathsf{h\ w} = \mathsf{g\ w}$$

Let $\mathsf{w} : \mathsf{B}$. By the bisimilarity version of the coinduction principle (Corollary 3.3.29), it is enough to prove $(\mathsf{Bisim}_i\ (\mathsf{h\ w})\ (\mathsf{g\ w}))$.[54]

In other words, we need to construct a bisimulation $\mathsf{R} : (\mathsf{A_F} \times \mathsf{A_F}) \to \mathcal{U}_i$ such that $\mathsf{R}\ (\mathsf{h\ w},\ \mathsf{g\ w})$ is inhabited.

Define R by Σ-recursion as (for any $\mathsf{a},\mathsf{b} : \mathsf{A_F}$):

$$\mathsf{R}\ (\mathsf{a},\mathsf{b}) :\equiv \sum_{\mathsf{c}:\mathsf{B}} (\mathsf{a} = \mathsf{h\ c}) \times (\mathsf{b} = \mathsf{g\ c})$$

Notice that $\mathsf{R}\ (\mathsf{a},\mathsf{b})$ is a type at level i, because B is at level $i-1$. Also, type $(\mathsf{a} = \mathsf{h\ c}) \times (\mathsf{b} = \mathsf{g\ c})$ is at level i, since $\mathsf{A_F}$ is at level i.

Then, $\mathsf{R}\ (\mathsf{h\ w},\ \mathsf{g\ w})$ is inhabited by:

$$(\ \mathsf{w},\ (\mathsf{refl_{A_F}}\ (\mathsf{h\ w}),\ \mathsf{refl_{A_F}}\ (\mathsf{g\ w}))\ ) : \mathsf{R}\ (\mathsf{h\ w},\ \mathsf{g\ w})$$

[53] We are only interested in the existence of a proof w_1, we do not require the specific term w_1 stands for.

[54] We are setting $m = i$ in Corollary 3.3.29.

It remains to show that R is a bisimulation. Let $\mathsf{m}, \mathsf{n} : \mathsf{A_F}$. We have to show that the following type is inhabited:

$$(\mathsf{R}\ (\mathsf{m}, \mathsf{n})) \to \left(\sum_{\mathsf{t}:\ \mathsf{H}} \sum_{\mathsf{z}:\mathsf{A_F}\times\mathsf{A_F}} \mathsf{R}\ \mathsf{z} \right) (\mathsf{Out}_{\mathsf{A_F}}\ \mathsf{m} = \mathsf{map}^{\mathsf{H}}\ (\mathsf{pr}_1 \circ \mathsf{pr}_1)\ \mathsf{t}) \times (\mathsf{Out}_{\mathsf{A_F}}\ \mathsf{n} = \mathsf{map}^{\mathsf{H}}\ (\mathsf{pr}_2 \circ \mathsf{pr}_1)\ \mathsf{t})$$

or equivalently, by definition of R and Σ-recursion:

$$\prod_{\mathsf{c}:\mathsf{B}} (\mathsf{m} = \mathsf{h}\ \mathsf{c}) \to (\mathsf{n} = \mathsf{g}\ \mathsf{c}) \to \left(\sum_{\mathsf{t}:\ \mathsf{H}} \sum_{\mathsf{z}:\mathsf{A_F}\times\mathsf{A_F}} \mathsf{R}\ \mathsf{z} \right) (\mathsf{Out}_{\mathsf{A_F}}\ \mathsf{m} = \mathsf{map}^{\mathsf{H}}\ (\mathsf{pr}_1 \circ \mathsf{pr}_1)\ \mathsf{t}) \times (\mathsf{Out}_{\mathsf{A_F}}\ \mathsf{n} = \mathsf{map}^{\mathsf{H}}\ (\mathsf{pr}_2 \circ \mathsf{pr}_1)\ \mathsf{t})$$

Let $\mathsf{c} : \mathsf{B}$ with the following inhabited types:

$$\mathsf{m} = \mathsf{h}\ \mathsf{c} \qquad \mathsf{n} = \mathsf{g}\ \mathsf{c} \tag{3.33}$$

First, define a function:

$$\mathsf{k} : \mathsf{B} \to \left(\sum_{\mathsf{z}:\mathsf{A_F}\times\mathsf{A_F}} \mathsf{R}\ \mathsf{z} \right)$$

as:

$$\mathsf{k}\ \mathsf{r} :\equiv (\ (\mathsf{h}\ \mathsf{r}, \mathsf{g}\ \mathsf{r}),\ (\mathsf{r},\ (\mathsf{refl}_{\mathsf{A_F}}\ (\mathsf{h}\ \mathsf{r}),\ \mathsf{refl}_{\mathsf{A_F}}\ (\mathsf{g}\ \mathsf{r}))\ )\ )$$

Now, define:

$$\mathsf{t} :\equiv \mathsf{map}^{\mathsf{H}}\ \mathsf{k}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{c}) : \mathsf{H} \sum_{\mathsf{z}:\mathsf{A_F}\times\mathsf{A_F}} \mathsf{R}\ \mathsf{z}$$

It remains to show that this t satisfies the two required equations. We have:

$$\begin{aligned}
\mathsf{map}^{\mathsf{H}}\ (\mathsf{pr}_1 \circ \mathsf{pr}_1)\ \mathsf{t} &\overset{(1)}{=} \mathsf{map}^{\mathsf{H}}\ (\mathsf{pr}_1 \circ \mathsf{pr}_1)\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{k}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{c})) \\
&\overset{(2)}{=} \mathsf{map}^{\mathsf{H}}\ (\mathsf{pr}_1 \circ \mathsf{pr}_1 \circ \mathsf{k})\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{c}) \\
&\overset{(3)}{=} \mathsf{map}^{\mathsf{H}}\ (\lambda \mathsf{r}.\ \mathsf{h}\ \mathsf{r})\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{c}) \\
&\overset{(4)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{h}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{c}) \\
&\overset{(5)}{=} \mathsf{Out}_{\mathsf{A_F}}\ (\mathsf{h}\ \mathsf{c}) \\
&\overset{(6)}{=} \mathsf{Out}_{\mathsf{A_F}}\ \mathsf{m}
\end{aligned}$$

where (1) follows by definition of t, (2) by functorial mapping of H, (3) by definition of function composition and function k,[55] (4) by the uniqueness rule for Π-types on page 21, (5) by commutativity of Diagram (3.31), and (6) holds because $\mathsf{m} = \mathsf{h}\ \mathsf{c}$ is inhabited by Equations (3.33).

Similarly:

$$\begin{aligned}
\mathsf{map}^{\mathsf{H}}\ (\mathsf{pr}_2 \circ \mathsf{pr}_1)\ \mathsf{t} &\overset{(1)}{=} \mathsf{map}^{\mathsf{H}}\ (\mathsf{pr}_2 \circ \mathsf{pr}_1)\ (\mathsf{map}^{\mathsf{H}}\ \mathsf{k}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{c})) \\
&\overset{(2)}{=} \mathsf{map}^{\mathsf{H}}\ (\mathsf{pr}_2 \circ \mathsf{pr}_1 \circ \mathsf{k})\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{c})
\end{aligned}$$

[55] In more detail, we have by computation:

$$\begin{aligned}
\mathsf{pr}_1 \circ \mathsf{pr}_1 \circ \mathsf{k} &\equiv \lambda \mathsf{r}.\ \mathsf{pr}_1\ (\mathsf{pr}_1\ (\mathsf{k}\ \mathsf{r})) \\
&\equiv \lambda \mathsf{r}.\ \mathsf{pr}_1\ (\mathsf{pr}_1\ (\ (\mathsf{h}\ \mathsf{r}, \mathsf{g}\ \mathsf{r}),\ (\mathsf{r},\ (\mathsf{refl}_{\mathsf{A_F}}\ (\mathsf{h}\ \mathsf{r}),\ \mathsf{refl}_{\mathsf{A_F}}\ (\mathsf{g}\ \mathsf{r}))\ )\ )) \\
&\equiv \lambda \mathsf{r}.\ \mathsf{pr}_1\ (\mathsf{h}\ \mathsf{r}, \mathsf{g}\ \mathsf{r}) \\
&\equiv \lambda \mathsf{r}.\ \mathsf{h}\ \mathsf{r}
\end{aligned}$$

$$\overset{(3)}{=} \mathsf{map}^{\mathsf{H}}\ (\lambda \mathsf{r}.\ \mathsf{g}\ \mathsf{r})\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{c})$$
$$\overset{(4)}{=} \mathsf{map}^{\mathsf{H}}\ \mathsf{g}\ (\mathsf{Out}_{\mathsf{B}}\ \mathsf{c})$$
$$\overset{(5)}{=} \mathsf{Out}_{\mathsf{A_F}}\ (\mathsf{g}\ \mathsf{c})$$
$$\overset{(6)}{=} \mathsf{Out}_{\mathsf{A_F}}\ \mathsf{n}$$

where (1) follows by definition of t, (2) by functorial mapping of H, (3) by definition of function composition and function k, (4) by the uniqueness rule for Π-types on page 21, (5) by commutativity of Diagram (3.32), and (6) holds because $\mathsf{n} = \mathsf{g}\ \mathsf{c}$ is inhabited by Equations (3.33).

□

Theorem 3.3.30 states that the problem of building a (lower level) final coalgebra is reduced to the problem of building a (lower level) weakly final coalgebra.

The theorem has the following corollary.

Corollary 3.3.31. *Let $i > 0$, and $\mathsf{H} : \mathcal{U}_i \to \mathcal{U}_i$ a functor that preserves sets. Then, H has a lower level final H-coalgebra. In other words, the following type is inhabited:*

$$\mathsf{LLFinCoAlg}_i\ \mathsf{H}$$

Proof. By Lemma 3.3.6, H has a lower level weakly final H-coalgebra at universe level i. Denote this LLWfinCoAlg by $(\mathsf{W}, \mathsf{Out}_{\mathsf{W}}, \mathsf{m})$.

But then, by Theorem 3.3.30(i), $(\mathsf{W_F}, \mathsf{Out}_{\mathsf{W_F}}, \mathsf{mc})$ is a lower level final H-coalgebra at universe level i.

□

Therefore, any set-preserving endofunctor has a *lower level* final H-coalgebra. What about a final H-coalgebra?

3.3.3 A functor without a final coalgebra

In Section 3.2.3, it was shown that the powertype functor $\mathsf{Pow}_{i,j}$ (see Definition 3.2.23 and Lemma 3.2.24) does not have an initial set $\mathsf{Pow}_{i,j}$-algebra (Theorem 3.2.25).

In this section, we will show that functor $\mathsf{Pow}_{i,j}$ does not have a final $\mathsf{Pow}_{i,j}$-coalgebra. However, by Corollary 3.3.31, it does have a lower level final $\mathsf{Pow}_{i,j}$-coalgebra, since $\mathsf{Pow}_{i,j}$ is a set-preserving endofunctor (Lemma 3.2.24).

Theorem 3.3.32. *Let $i > 0$ and $j < i$ be two universe levels. The functor $\mathsf{Pow}_{i,j}$ does not have a final $\mathsf{Pow}_{i,j}$-coalgebra. In other words, the following type is inhabited:*

$$\neg(\mathsf{FinCoAlg}_i\ \mathsf{Pow}_{i,j})$$

or, equivalently, type $\mathsf{FinCoAlg}_i\ \mathsf{Pow}_{i,j}$ is not *inhabited.*

Proof. Suppose, for a contradiction, that $(\mathsf{A}, \mathsf{Out}_{\mathsf{A}}, \mathsf{mc}) : \mathsf{FinCoAlg}_i\ \mathsf{Pow}_{i,j}$ is a final $\mathsf{Pow}_{i,j}$-coalgebra.

Since $\mathsf{Pow}_{i,j}$ is an endofunctor (Lemma 3.2.24), we can apply Lambek's Lemma (Lemma 3.3.10) to obtain that $\mathsf{Pow}_{i,j}\ \mathsf{A}$ and A are equivalent types.

Therefore, by definition of $\mathsf{Pow}_{i,j}$, types $\mathsf{A} \to \mathbf{Prop}_j$ and A are equivalent, which is impossible by Lemma 1.8.20.

□

Theorem 3.3.32 implies that any LLFinCoAlg for functor $\mathsf{Pow}_{i,j}$ is not a FinCoAlg. In particular, the LLFinCoAlg constructed by Corollary 3.3.31, is not a FinCoAlg for functor $\mathsf{Pow}_{i,j}$. Hence, not every LLFinCoAlg is a FinCoAlg.

We also have a corollary regarding the non-existence of weakly final coalgebras for functor $\mathsf{Pow}_{i,j}$.

Corollary 3.3.33. *Let $i > 0$ and $j < i$ be two universe levels. The functor $\mathsf{Pow}_{i,j}$ does not have a weakly final $\mathsf{Pow}_{i,j}$-coalgebra. In other words, the following type is inhabited:*

$$\neg(\mathsf{WfinCoAlg}_i\ \mathsf{Pow}_{i,j})$$

or, equivalently, type $\mathsf{WfinCoAlg}_i\ \mathsf{Pow}_{i,j}$ is not *inhabited.*

Proof. Suppose $(\mathsf{A}, \mathsf{Out}_\mathsf{A}, \mathsf{m}) : \mathsf{WfinCoAlg}_i\ \mathsf{Pow}_{i,j}$.

By Theorem 3.3.30(ii), $(\mathsf{A}_\mathsf{F}, \mathsf{Out}_{\mathsf{A}_\mathsf{F}}, \mathsf{mc})$ is a final $\mathsf{Pow}_{i,j}$-coalgebra, which contradicts Theorem 3.3.32. □

Another corollary states that no LLFinCoAlg for $\mathsf{Pow}_{i,j}$ satisfies the conclusion of Lambek's Lemma. In other words, a LLFinCoAlg will not be, in general, a "fixpoint" for its functor.

Corollary 3.3.34. *Let $i > 0$ and $j < i$ be two universe levels, and $(\mathsf{A}, \mathsf{Out}_\mathsf{A}, \mathsf{mc})$ a lower level final $\mathsf{Pow}_{i,j}$-coalgebra. Then, $\mathsf{Pow}_{i,j}\ \mathsf{A}$ and A are not equivalent types. In other words, the following type is inhabited:*

$$\neg(\mathsf{Pow}_{i,j}\ \mathsf{A} \simeq \mathsf{A})$$

In particular, the LLFinCoAlg constructed by Corollary 3.3.31, is not a "fixpoint" for functor $\mathsf{Pow}_{i,j}$.

Proof. Let $(\mathsf{A}, \mathsf{Out}_\mathsf{A}, \mathsf{mc}) : \mathsf{LLFinCoAlg}_i\ \mathsf{Pow}_{i,j}$. Suppose, for a contradiction, that $\mathsf{Pow}_{i,j}\ \mathsf{A}$ and A are equivalent types.

Then, by definition of $\mathsf{Pow}_{i,j}$, types $\mathsf{A} \to \mathbf{Prop}_j$ and A are equivalent, which is impossible by Lemma 1.8.20. □

These results support the claim that LLFinCoAlgs are weaker than FinCoAlgs. Nevertheless, LLFinCoAlgs are still *useful*, as they can define *unique* functions by corecursive equations (see Section 3.3.4).

And again, these results emphasized the importance of explicitly handling universe levels. Ignoring them produces the *apparent* contradiction that every set-preserving endofunctor has a final coalgebra, while at the same time the $\mathsf{Pow}_{i,j}$ functor does not have one.

We proceed to show an example on how these lower level final H-coalgebras are used.

3.3.4 Example: Streams

We will construct the streams as a LLFinCoAlg. First, we need to define the functor.

Definition 3.3.35. Let $\mathsf{A} : \mathcal{U}_i$ be a type. We define function $\mathsf{StreamF}_i\ \mathsf{A} : \mathcal{U}_i \to \mathcal{U}_i$ as:

$$\mathsf{StreamF}_i\ \mathsf{A} :\equiv (\lambda \mathsf{Y} : \mathcal{U}_i.\ \mathsf{A} \times \mathsf{Y}) : \mathcal{U}_i \to \mathcal{U}_i$$

Given $\mathsf{C}, \mathsf{D} : \mathcal{U}_i$ and function $\mathsf{f} : \mathsf{C} \to \mathsf{D}$, we define the mapping function as:

$$\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{f}\ (\mathsf{o}, \mathsf{s}) :\equiv (\mathsf{o}, \mathsf{f}\ \mathsf{s})$$

having type:

$$\prod_{\mathsf{C},\mathsf{D}:\mathcal{U}_i} (\mathsf{C} \to \mathsf{D}) \to ((\mathsf{A} \times \mathsf{C}) \to (\mathsf{A} \times \mathsf{D}))$$

▲

Function $\mathsf{StreamF}_i\ \mathsf{A} : \mathcal{U}_i \to \mathcal{U}_i$ is well-typed, as we explain next. Given $\mathsf{Y} : \mathcal{U}_i$, type $\mathsf{A} \times \mathsf{Y}$ is at level i, because type A is at level i and the product type is at level $\max\{i, i\} = i$ (see Remark 1.5.29).

Function $\mathsf{StreamF}_i\ \mathsf{A}$ and its mapping function form a functor.

Lemma 3.3.36. *Let $\mathsf{A} : \mathcal{U}_i$ be a type. The function $\mathsf{StreamF}_i\ \mathsf{A}$ (together with its mapping function $\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}$) is an endofunctor. If in addition A is a set, then $\mathsf{StreamF}_i\ \mathsf{A}$ also preserves sets.*

Proof. It is obvious from its definition that $\mathsf{StreamF}_i\ \mathsf{A}$ is an endofunctor. When A is a set, $\mathsf{StreamF}_i\ \mathsf{A}$ preserves sets because sets are closed under Σ-types (hence, product types), by Lemma 1.8.18.

It remains to show the two functorial properties (see Definition 1.9.6).

- Type $\prod_{\mathsf{C}:\mathcal{U}_i} \prod_{\mathsf{w}:\mathsf{A}\times\mathsf{C}} \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{id}_\mathsf{C}\ \mathsf{w} = \mathsf{w}$ is inhabited.

  Let $\mathsf{C} : \mathcal{U}_i$. By the induction principle for Σ-types (Lemma 1.5.24), it is enough to prove:

$$\prod_{\mathsf{a}:\mathsf{A}} \prod_{\mathsf{c}:\mathsf{C}} \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{id}_\mathsf{C}\ (\mathsf{a}, \mathsf{c}) = (\mathsf{a}, \mathsf{c})$$

or equivalently, by definition of $\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}$:

$$\prod_{\mathsf{a}:\mathsf{A}} \prod_{\mathsf{c}:\mathsf{C}} (\mathsf{a}, \mathsf{id}_{\mathsf{C}}\ \mathsf{c}) = (\mathsf{a}, \mathsf{c})$$

which is trivially inhabited by:

$$\lambda(\mathsf{a}:\mathsf{A})(\mathsf{c}:\mathsf{C}).\ \mathsf{refl}_{(\mathsf{A}\times\mathsf{C})}\ (\mathsf{a}, \mathsf{c})$$

- The following type is inhabited:

$$\prod_{\mathsf{B},\mathsf{C},\mathsf{D}:\mathcal{U}_i} \prod_{\mathsf{g}:\mathsf{B}\to\mathsf{C}} \prod_{\mathsf{h}:\mathsf{C}\to\mathsf{D}} \prod_{\mathsf{w}:\mathsf{A}\times\mathsf{B}} \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{h}\circ\mathsf{g})\ \mathsf{w} = \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{h}\ (\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{g}\ \mathsf{w})$$

Let $\mathsf{B}, \mathsf{C}, \mathsf{D} : \mathcal{U}_i$, $\mathsf{g} : \mathsf{B} \to \mathsf{C}$, and $\mathsf{h} : \mathsf{C} \to \mathsf{D}$. Again, by the induction principle for Σ-types, it is enough to prove:

$$\prod_{\mathsf{a}:\mathsf{A}} \prod_{\mathsf{b}:\mathsf{B}} \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{h}\circ\mathsf{g})\ (\mathsf{a}, \mathsf{b}) = \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{h}\ (\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{g}\ (\mathsf{a}, \mathsf{b}))$$

or equivalently, by definition of $\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}$:

$$\prod_{\mathsf{a}:\mathsf{A}} \prod_{\mathsf{b}:\mathsf{B}} (\mathsf{a}, \mathsf{h}\ (\mathsf{g}\ \mathsf{b})) = \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{h}\ (\mathsf{a}, \mathsf{g}\ \mathsf{b})$$

which simplifies to:

$$\prod_{\mathsf{a}:\mathsf{A}} \prod_{\mathsf{b}:\mathsf{B}} (\mathsf{a}, \mathsf{h}\ (\mathsf{g}\ \mathsf{b})) = (\mathsf{a}, \mathsf{h}\ (\mathsf{g}\ \mathsf{b}))$$

and is trivially inhabited by:

$$\lambda(\mathsf{a}:\mathsf{A})(\mathsf{b}:\mathsf{B}).\ \mathsf{refl}_{(\mathsf{A}\times\mathsf{D})}\ (\mathsf{a}, \mathsf{h}\ (\mathsf{g}\ \mathsf{b}))$$

□

Therefore, when $i > 0$ and A is a set, functor $\mathsf{StreamF}_i\ \mathsf{A}$ has a LLFinCoAlg, by Corollary 3.3.31. We introduce notation for this.

Definition 3.3.37. Let $i > 0$ be a universe level and $\mathsf{A} : \mathbf{Set}_i$ a set. We will denote by $(\mathsf{Stream}_i\ \mathsf{A}, \mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}, \mathsf{mc})$ the lower level final $\mathsf{StreamF}_i\ \mathsf{A}$-coalgebra constructed by Corollary 3.3.31.

▲

Observe that the $\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}$ function has type:

$$(\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{StreamF}_i\ \mathsf{A}\ (\mathsf{Stream}_i\ \mathsf{A}))$$

which is equivalent to:

$$(\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{A} \times (\mathsf{Stream}_i\ \mathsf{A}))$$

Intuitively, this means we can "split" the $\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}$ function into two functions $(\mathsf{Stream}_i\ \mathsf{A}) \to \mathsf{A}$ and $(\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_i\ \mathsf{A})$, which will be the destructors for $\mathsf{Stream}_i\ \mathsf{A}$. These destructors correspond to the head and tail functions, respectively.

Definition 3.3.38. Let $i > 0$ be a universe level and $\mathsf{A} : \mathbf{Set}_i$ a set. We define two *destructors* for $\mathsf{Stream}_i\ \mathsf{A}$.

- The *head function* $\overline{\mathsf{head}}_i : (\mathsf{Stream}_i\ \mathsf{A}) \to \mathsf{A}$ is defined as:[56]

$$\overline{\mathsf{head}}_i :\equiv \lambda\mathsf{s}:\mathsf{Stream}_i\ \mathsf{A}.\ \mathsf{pr}_1\ (\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{s})$$

[56] We will write $\overline{\mathsf{head}}_i$, instead of $\overline{\mathsf{head}}_i\ \mathsf{A}$, leaving A implicit. But on those rare cases when we want to emphasize A, we will write $\overline{\mathsf{head}}_i^{\mathsf{A}}$. The tail function follows this convention as well.

- The *tail function* $\overline{\mathsf{tail}}_i : (\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_i\ \mathsf{A})$ is defined as:

$$\overline{\mathsf{tail}}_i :\equiv \lambda \mathsf{s} : \mathsf{Stream}_i\ \mathsf{A}.\ \mathsf{pr}_2\ (\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{s})$$

▲

Next, we prove a coinduction principle for $\mathsf{Stream}_i\ \mathsf{A}$. It essentially expresses that two terms in $\mathsf{Stream}_i\ \mathsf{A}$ are identifiable if there is a *stream bisimulation* that relates them.

Theorem 3.3.39 (Coinduction principle for $\mathsf{Stream}_i\ \mathsf{A}$)**.** *Let $i > 0$ and $m \leq i$ be two universe levels. Let $\mathsf{A} : \mathbf{Set}_i$ be a set, and $\mathsf{w}, \mathsf{y} : \mathsf{Stream}_i\ \mathsf{A}$ two terms.*

If the following type is inhabited:

$$\sum_{\mathsf{R}:(\mathsf{Stream}_i\ \mathsf{A})\to(\mathsf{Stream}_i\ \mathsf{A})\to\mathcal{U}_m} (\mathsf{R}\ \mathsf{w}\ \mathsf{y}) \times \left(\prod_{\mathsf{m},\mathsf{n}:\mathsf{Stream}_i\ \mathsf{A}} (\mathsf{R}\ \mathsf{m}\ \mathsf{n}) \to \left((\overline{\mathsf{head}}_i\ \mathsf{m} = \overline{\mathsf{head}}_i\ \mathsf{n}) \times (\mathsf{R}\ (\overline{\mathsf{tail}}_i\ \mathsf{m})\ (\overline{\mathsf{tail}}_i\ \mathsf{n})) \right) \right)$$

then, the following type is inhabited:

$$\mathsf{w} = \mathsf{y}$$

Proof. Let us suppose we have some $\mathsf{R} : (\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_i\ \mathsf{A}) \to \mathcal{U}_m$ such that $\mathsf{h}_1 : \mathsf{R}\ \mathsf{w}\ \mathsf{y}$ and:

$$\mathsf{h}_2 : \prod_{\mathsf{m},\mathsf{n}:\mathsf{Stream}_i\ \mathsf{A}} (\mathsf{R}\ \mathsf{m}\ \mathsf{n}) \to \left((\overline{\mathsf{head}}_i\ \mathsf{m} = \overline{\mathsf{head}}_i\ \mathsf{n}) \times (\mathsf{R}\ (\overline{\mathsf{tail}}_i\ \mathsf{m})\ (\overline{\mathsf{tail}}_i\ \mathsf{n})) \right)$$

We are going to use the bisimilarity coinduction principle (Corollary 3.3.29) to prove that $\mathsf{w} = \mathsf{y}$ is inhabited.[57]

Therefore, we need to prove that $(\mathsf{Bisim}_m\ \mathsf{w}\ \mathsf{y})$ is inhabited.

This means we need to define some bisimulation $\mathsf{S} : ((\mathsf{Stream}_i\ \mathsf{A}) \times (\mathsf{Stream}_i\ \mathsf{A})) \to \mathcal{U}_m$ such that $\mathsf{S}\ (\mathsf{w}, \mathsf{y})$ is inhabited.

Define:

$$\mathsf{S} :\equiv \lambda \mathsf{p}.\ \mathsf{R}\ (\mathsf{pr}_1\ \mathsf{p})\ (\mathsf{pr}_2\ \mathsf{p})$$

We have $\mathsf{S}\ (\mathsf{w}, \mathsf{y}) \equiv \mathsf{R}\ \mathsf{w}\ \mathsf{y}$ by definition, hence $\mathsf{h}_1 : \mathsf{S}\ (\mathsf{w}, \mathsf{y})$. It remains to show that S is a bisimulation.

Let $\mathsf{m}, \mathsf{n} : \mathsf{Stream}_i\ \mathsf{A}$ with $\mathsf{p}_1 : \mathsf{S}\ (\mathsf{m}, \mathsf{n})$ (which simplifies to $\mathsf{p}_1 : \mathsf{R}\ \mathsf{m}\ \mathsf{n}$). Then we have:

$$\mathsf{p}_2 :\equiv \mathsf{pr}_1\ (\mathsf{h}_2\ \mathsf{m}\ \mathsf{n}\ \mathsf{p}_1) : \overline{\mathsf{head}}_i\ \mathsf{m} = \overline{\mathsf{head}}_i\ \mathsf{n}$$
$$\mathsf{p}_3 :\equiv \mathsf{pr}_2\ (\mathsf{h}_2\ \mathsf{m}\ \mathsf{n}\ \mathsf{p}_1) : \mathsf{R}\ (\overline{\mathsf{tail}}_i\ \mathsf{m})\ (\overline{\mathsf{tail}}_i\ \mathsf{n})$$

where p_3 also has type $\mathsf{S}\ (\overline{\mathsf{tail}}_i\ \mathsf{m},\ \overline{\mathsf{tail}}_i\ \mathsf{n})$ by definition of S.

To finish the proof, we have to construct a term of type:

$$\sum_{\mathsf{t}:\ \mathsf{A}\times\left(\sum_{\mathsf{z}:(\mathsf{Stream}_i\ \mathsf{A})\times(\mathsf{Stream}_i\ \mathsf{A})} \mathsf{S}\ \mathsf{z}\right)} (\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{m} = \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{pr}_1 \circ \mathsf{pr}_1)\ \mathsf{t}) \times (\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{n} = \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{pr}_2 \circ \mathsf{pr}_1)\ \mathsf{t})$$

Define:

$$\mathsf{t} :\equiv (\ \overline{\mathsf{head}}_i\ \mathsf{m},\ (\ (\overline{\mathsf{tail}}_i\ \mathsf{m}, \overline{\mathsf{tail}}_i\ \mathsf{n}),\ \mathsf{p}_3\ )\ ) : \mathsf{A} \times \left(\sum_{\mathsf{z}:(\mathsf{Stream}_i\ \mathsf{A})\times(\mathsf{Stream}_i\ \mathsf{A})} \mathsf{S}\ \mathsf{z} \right)$$

It remains to show that this term t satisfies the two required equations. We have:

$$\begin{aligned}
\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{pr}_1 \circ \mathsf{pr}_1)\ \mathsf{t} &\overset{(1)}{=} (\ \overline{\mathsf{head}}_i\ \mathsf{m},\ \mathsf{pr}_1\ (\mathsf{pr}_1\ (\ (\overline{\mathsf{tail}}_i\ \mathsf{m}, \overline{\mathsf{tail}}_i\ \mathsf{n}),\ \mathsf{p}_3\ ))\ ) \\
&\overset{(2)}{=} (\ \overline{\mathsf{head}}_i\ \mathsf{m},\ \overline{\mathsf{tail}}_i\ \mathsf{m}\ ) \\
&\overset{(3)}{=} (\ \mathsf{pr}_1\ (\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{m}),\ \mathsf{pr}_2\ (\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{m})\ ) \\
&\overset{(4)}{=} \mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{m}
\end{aligned}$$

[57] Notice that we can use this coinduction principle, because $\mathsf{Stream}_i\ \mathsf{A}$ has the form W_F, where W is the LLWfinCoAlg constructed by Lemma 3.3.6.

where (1) follows by definition of t and $\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}$, (2) by definition of pr_1, (3) by definition of $\overline{\mathsf{head}}_i$ and $\overline{\mathsf{tail}}_i$, and (4) by Lemma 1.6.1.

Similarly:

$$\begin{aligned}
\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{pr}_2 \circ \mathsf{pr}_1)\ \mathsf{t} &\overset{(1)}{=} (\ \overline{\mathsf{head}}_i\ \mathsf{m},\ \mathsf{pr}_2\ (\mathsf{pr}_1\ (\ (\overline{\mathsf{tail}}_i\ \mathsf{m}, \overline{\mathsf{tail}}_i\ \mathsf{n}),\ \mathsf{p}_3\ ))\ ) \\
&\overset{(2)}{=} (\ \overline{\mathsf{head}}_i\ \mathsf{m},\ \overline{\mathsf{tail}}_i\ \mathsf{n}\ ) \\
&\overset{(3)}{=} (\ \overline{\mathsf{head}}_i\ \mathsf{n},\ \overline{\mathsf{tail}}_i\ \mathsf{n}\ ) \\
&\overset{(4)}{=} (\ \mathsf{pr}_1\ (\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{n}),\ \mathsf{pr}_2\ (\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{n})\ ) \\
&\overset{(5)}{=} \mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{n}
\end{aligned}$$

where (1) follows by definition of t and $\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}$, (2) by definition of pr_1 and pr_2, (3) holds because $\overline{\mathsf{head}}_i\ \mathsf{m} = \overline{\mathsf{head}}_i\ \mathsf{n}$ is inhabited, (4) by definition of $\overline{\mathsf{head}}_i$ and $\overline{\mathsf{tail}}_i$, and (5) by Lemma 1.6.1.

□

Next, we prove a stream coiteration principle.

Theorem 3.3.40 (Coiteration principle for $\mathsf{Stream}_i\ \mathsf{A}$)**.** *Let $i > 0$ be a universe level and $\mathsf{A} : \mathbf{Set}_i$ a set. Let $\mathsf{D} : \mathcal{U}_{i-1}$ be a type, $\mathsf{h} : \mathsf{D} \to \mathsf{A}$ a function, and $\mathsf{t} : \mathsf{D} \to \mathsf{D}$ a function. Then, the following type is inhabited:*

$$\mathsf{IsContr}\left[\sum_{\mathsf{f}:\mathsf{D}\to(\mathsf{Stream}_i\ \mathsf{A})}\ \prod_{\mathsf{w}:\mathsf{D}} (\overline{\mathsf{head}}_i\ (\mathsf{f}\ \mathsf{w}) = \mathsf{h}\ \mathsf{w}) \times (\overline{\mathsf{tail}}_i\ (\mathsf{f}\ \mathsf{w}) = \mathsf{f}\ (\mathsf{t}\ \mathsf{w}))\right] \tag{3.34}$$

In other words, there is a unique *function $\mathsf{f} : \mathsf{D} \to (\mathsf{Stream}_i\ \mathsf{A})$ satisfying the stated corecursive equations.*

Proof. First, define a function $\mathsf{Out}_\mathsf{D} : \mathsf{D} \to (\mathsf{StreamF}_i\ \mathsf{A}\ \mathsf{D})$, or equivalently, $\mathsf{Out}_\mathsf{D} : \mathsf{D} \to (\mathsf{A} \times \mathsf{D})$ as follows:

$$\mathsf{Out}_\mathsf{D}\ \mathsf{d} :\equiv (\mathsf{h}\ \mathsf{d}, \mathsf{t}\ \mathsf{d})$$

This means that $(\mathsf{D}, \mathsf{Out}_\mathsf{D})$ is a $\mathsf{StreamF}_i\ \mathsf{A}$-coalgebra at level $i-1$. Therefore, since $\mathsf{Stream}_i\ \mathsf{A}$ is a LLFinCoAlg, there is a *unique* morphism $\mathsf{u} : \mathsf{D} \to (\mathsf{Stream}_i\ \mathsf{A})$ making the following diagram commute:

$$\begin{array}{ccc}
\mathsf{D} & \xrightarrow{\quad\mathsf{u}\quad} & \mathsf{Stream}_i\ \mathsf{A} \\
{\scriptstyle \mathsf{Out}_\mathsf{D}}\downarrow & & \downarrow{\scriptstyle \mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}} \\
\mathsf{A} \times \mathsf{D} & \xrightarrow[\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{u}]{} & \mathsf{A} \times (\mathsf{Stream}_i\ \mathsf{A})
\end{array} \tag{3.35}$$

To prove that type (3.34) is inhabited (see Definition 1.8.1), we have to construct a function f satisfying the stated equations (i.e. the existence step), such that any other function satisfying those equations is equal to f (i.e. the uniqueness step).

Existence

We propose the morphism u above as our candidate function. It remains to construct a proof w_1 that u satisfies the equations in type (3.34).

Let $\mathsf{w} : \mathsf{D}$. First, we prove the following identities:

$$\begin{aligned}
\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ (\mathsf{u}\ \mathsf{w}) &\overset{(1)}{=} \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{u}\ (\mathsf{Out}_\mathsf{D}\ \mathsf{w}) \\
&\overset{(2)}{=} \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{u}\ (\mathsf{h}\ \mathsf{w}, \mathsf{t}\ \mathsf{w}) \\
&\overset{(3)}{=} (\mathsf{h}\ \mathsf{w}, \mathsf{u}\ (\mathsf{t}\ \mathsf{w}))
\end{aligned}$$

where (1) follows by commutativity of Diagram (3.35), (2) by definition of Out_D, and (3) by definition of $\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}$.

Therefore, by applying pr_1 and pr_2 to both sides of the above identity, we get:

$$\overline{\mathsf{head}}_i\ (\mathsf{u}\ \mathsf{w}) \overset{(1)}{=} \mathsf{pr}_1\ (\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ (\mathsf{u}\ \mathsf{w})) = \mathsf{pr}_1\ (\mathsf{h}\ \mathsf{w}, \mathsf{u}\ (\mathsf{t}\ \mathsf{w})) = \mathsf{h}\ \mathsf{w}$$
$$\overline{\mathsf{tail}}_i\ (\mathsf{u}\ \mathsf{w}) \overset{(2)}{=} \mathsf{pr}_2\ (\mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}\ (\mathsf{u}\ \mathsf{w})) = \mathsf{pr}_2\ (\mathsf{h}\ \mathsf{w}, \mathsf{u}\ (\mathsf{t}\ \mathsf{w})) = \mathsf{u}\ (\mathsf{t}\ \mathsf{w})$$

where (1) follows by definition of $\overline{\mathsf{head}}_i$, and (2) by definition of $\overline{\mathsf{tail}}_i$.

Uniqueness

By definition of contractible type, we have to prove that the following type is inhabited:

$$\prod_{(\mathsf{g},\mathsf{w}_2):\sum_{\mathsf{f}:\mathsf{D}\to(\mathsf{Stream}_i\ \mathsf{A})}\prod_{\mathsf{w}:\mathsf{D}}(\overline{\mathsf{head}}_i\ (\mathsf{f}\ \mathsf{w})=\mathsf{h}\ \mathsf{w})\times(\overline{\mathsf{tail}}_i\ (\mathsf{f}\ \mathsf{w})=\mathsf{f}\ (\mathsf{t}\ \mathsf{w}))} (\mathsf{u}, \mathsf{w}_1) = (\mathsf{g}, \mathsf{w}_2)$$

where w_1 is a proof (constructed in the existence part) that u satisfies the equations, and w_2 is a proof that g satisfies the equations.

So, let $\mathsf{g}\colon \mathsf{D} \to (\mathsf{Stream}_i\ \mathsf{A})$ satisfying the corecursive equations:

$$\prod_{\mathsf{w}:\mathsf{D}} (\overline{\mathsf{head}}_i\ (\mathsf{g}\ \mathsf{w}) = \mathsf{h}\ \mathsf{w}) \times (\overline{\mathsf{tail}}_i\ (\mathsf{g}\ \mathsf{w}) = \mathsf{g}\ (\mathsf{t}\ \mathsf{w})) \tag{3.36}$$

The second component of the sigma type:

$$\sum_{\mathsf{f}:\mathsf{D}\to(\mathsf{Stream}_i\ \mathsf{A})} \prod_{\mathsf{w}:\mathsf{D}} (\overline{\mathsf{head}}_i\ (\mathsf{f}\ \mathsf{w}) = \mathsf{h}\ \mathsf{w}) \times (\overline{\mathsf{tail}}_i\ (\mathsf{f}\ \mathsf{w}) = \mathsf{f}\ (\mathsf{t}\ \mathsf{w}))$$

(i.e. the corecursive equations) is a mere proposition for every $\mathsf{f}\colon \mathsf{D} \to (\mathsf{Stream}_i\ \mathsf{A})$, as we explain next. The identity $\overline{\mathsf{head}}_i\ (\mathsf{f}\ \mathsf{w}) = \mathsf{h}\ \mathsf{w}$ is a mere proposition, because it is defined on *set* A. The identity $\overline{\mathsf{tail}}_i\ (\mathsf{f}\ \mathsf{w}) = \mathsf{f}\ (\mathsf{t}\ \mathsf{w})$ is a mere proposition, because it is defined on *set* $\mathsf{Stream}_i\ \mathsf{A}$.[58] Also, by Lemma 1.8.17, propositions are closed under the product type and Π-types.

Therefore, by Lemma 1.8.14, to prove that $(\mathsf{u}, \mathsf{w}_1) = (\mathsf{g}, \mathsf{w}_2)$ is inhabited, it is enough to prove that $\mathsf{u} = \mathsf{g}$ is inhabited.

To prove $\mathsf{u} = \mathsf{g}$, we are going to show that g is a coalgebra morphism from D to $\mathsf{Stream}_i\ \mathsf{A}$, because u is the *only* such morphism by definition of LLFinCoAlg.

So, we need to construct a proof q_2 that the following diagram commutes:

$$\begin{array}{ccc} \mathsf{D} & \xrightarrow{\quad\mathsf{g}\quad} & \mathsf{Stream}_i\ \mathsf{A} \\ {\scriptstyle \mathsf{Out}_{\mathsf{D}}}\downarrow & & \downarrow{\scriptstyle \mathsf{Out}_{(\mathsf{Stream}_i\ \mathsf{A})}} \\ \mathsf{A}\times\mathsf{D} & \xrightarrow[\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{g}]{} & \mathsf{A}\times(\mathsf{Stream}_i\ \mathsf{A}) \end{array}$$

Let $\mathsf{w}\colon \mathsf{D}$. We are trying to prove the identity:

$$\mathsf{Out}_{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{g}\ \mathsf{w}) = \mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{g}\ (\mathsf{Out}_{\mathsf{D}}\ \mathsf{w})$$

which is on the product type $\mathsf{A} \times (\mathsf{Stream}_i\ \mathsf{A})$. So, by Corollary 1.6.26, it is enough to prove that the following two types are inhabited:

$$\mathsf{pr}_1\ (\mathsf{Out}_{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{g}\ \mathsf{w})) = \mathsf{pr}_1\ (\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{g}\ (\mathsf{Out}_{\mathsf{D}}\ \mathsf{w}))$$
$$\mathsf{pr}_2\ (\mathsf{Out}_{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{g}\ \mathsf{w})) = \mathsf{pr}_2\ (\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{g}\ (\mathsf{Out}_{\mathsf{D}}\ \mathsf{w}))$$

We have:

$$\mathsf{pr}_1\ (\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{g}\ (\mathsf{Out}_{\mathsf{D}}\ \mathsf{w})) \overset{(1)}{=} \mathsf{pr}_1\ (\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{g}\ (\mathsf{h}\ \mathsf{w}, \mathsf{t}\ \mathsf{w}))$$

[58] Remember that $\mathsf{Stream}_i\ \mathsf{A}$ has the form W_{F}, which is a quotient. Quotients are always sets, see Lemma 1.11.24.

$$\overset{(2)}{=} \mathsf{pr}_1\ (\mathsf{h}\ \mathsf{w}, \mathsf{g}\ (\mathsf{t}\ \mathsf{w}))$$
$$\overset{(3)}{=} \mathsf{h}\ \mathsf{w}$$
$$\overset{(4)}{=} \overline{\mathsf{head}}_i\ (\mathsf{g}\ \mathsf{w})$$
$$\overset{(5)}{=} \mathsf{pr}_1\ (\mathsf{Out}_{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{g}\ \mathsf{w}))$$

where (1) follows by definition of $\mathsf{Out}_{\mathsf{D}}$, (2) by definition of $\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}$, (3) by definition of pr_1, (4) by equations (3.36), and (5) by definition of $\overline{\mathsf{head}}_i$.

Similarly:

$$\mathsf{pr}_2\ (\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{g}\ (\mathsf{Out}_{\mathsf{D}}\ \mathsf{w})) \overset{(1)}{=} \mathsf{pr}_2\ (\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}\ \mathsf{g}\ (\mathsf{h}\ \mathsf{w}, \mathsf{t}\ \mathsf{w}))$$
$$\overset{(2)}{=} \mathsf{pr}_2\ (\mathsf{h}\ \mathsf{w}, \mathsf{g}\ (\mathsf{t}\ \mathsf{w}))$$
$$\overset{(3)}{=} \mathsf{g}\ (\mathsf{t}\ \mathsf{w})$$
$$\overset{(4)}{=} \overline{\mathsf{tail}}_i\ (\mathsf{g}\ \mathsf{w})$$
$$\overset{(5)}{=} \mathsf{pr}_2\ (\mathsf{Out}_{(\mathsf{StreamF}_i\ \mathsf{A})}\ (\mathsf{g}\ \mathsf{w}))$$

where (1) follows by definition of $\mathsf{Out}_{\mathsf{D}}$, (2) by definition of $\mathsf{map}^{(\mathsf{StreamF}_i\ \mathsf{A})}$, (3) by definition of pr_2, (4) by equations (3.36), and (5) by definition of $\overline{\mathsf{tail}}_i$.

□

Notice that Theorem 3.3.40 requires D to be a type at level $i-1$. This level restriction is forced by the fact that $\mathsf{Stream}_i\ \mathsf{A}$ is a LLFinCoAlg. As it happened with the natural numbers in Section 3.2.4, this universe level restriction forces us to play with universe levels whenever we attempt to define functions between streams.

For example, let us define a stream map function. The function takes as input a function f and a stream s, and it produces as output a stream where function f is applied on each element of s.

In other words, given $i > 0$, two sets $\mathsf{A}, \mathsf{B} : \mathbf{Set}_i$, and a function $\mathsf{f} : \mathsf{A} \to \mathsf{B}$, we want to define a function:

$$\mathsf{mapStr}_i\ \mathsf{f} : (\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_i\ \mathsf{B})$$

However, if we want to use the coiteration principle for $\mathsf{Stream}_i\ \mathsf{B}$, the domain level of this function needs to be lower than the codomain level. Therefore, this function needs to have type $(\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_{i+1}\ \mathsf{B})$.

Now, we have the functions:

$$\mathsf{f} \circ \overline{\mathsf{head}}_i^{\mathsf{A}} : (\mathsf{Stream}_i\ \mathsf{A}) \to \mathsf{B}$$
$$\overline{\mathsf{tail}}_i^{\mathsf{A}} : (\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_i\ \mathsf{A})$$

Therefore, by the coiteration principle for $(\mathsf{Stream}_{i+1}\ \mathsf{B})$, there is a *unique* function $\mathsf{mapStr}_i\ \mathsf{f} : (\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_{i+1}\ \mathsf{B})$ defined by the corecursive equations (for every $\mathsf{w} : \mathsf{Stream}_i\ \mathsf{A}$):

$$\begin{aligned} \overline{\mathsf{head}}_{i+1}^{\mathsf{B}}\ (\mathsf{mapStr}_i\ \mathsf{f}\ \mathsf{w}) &= \mathsf{f}\ (\overline{\mathsf{head}}_i^{\mathsf{A}}\ \mathsf{w}) \\ \overline{\mathsf{tail}}_{i+1}^{\mathsf{B}}\ (\mathsf{mapStr}_i\ \mathsf{f}\ \mathsf{w}) &= \mathsf{mapStr}_i\ \mathsf{f}\ (\overline{\mathsf{tail}}_i^{\mathsf{A}}\ \mathsf{w}) \end{aligned} \tag{3.37}$$

In other words, if we represent streams as infinite lists[59] $(\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \ldots)$, where:

$$\overline{\mathsf{head}}_i^{\mathsf{A}}\ (\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \ldots) \equiv \mathsf{a}_1$$
$$\overline{\mathsf{tail}}_i^{\mathsf{A}}\ (\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \ldots) \equiv (\mathsf{a}_2, \mathsf{a}_3, \ldots)$$

then, $\mathsf{mapStr}_i\ \mathsf{f}\ (\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \ldots)$ will be a stream having as head:

$$\mathsf{f}\ (\overline{\mathsf{head}}_i^{\mathsf{A}}\ (\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \ldots)) \equiv \mathsf{f}\ \mathsf{a}_1$$

[59] Bear in mind this is just informal notation to explain the intuition behind equations (3.37).

and tail:

$$\mathsf{mapStr}_i\ \mathsf{f}\ (\overline{\mathsf{tail}}_i^{\mathsf{A}}\ (\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \ldots)) \equiv \mathsf{mapStr}_i\ \mathsf{f}\ (\mathsf{a}_2, \mathsf{a}_3, \ldots)$$

which it turn will be a stream having as head and tail:

$$\mathsf{f}\ (\overline{\mathsf{head}}_i^{\mathsf{A}}\ (\mathsf{a}_2, \mathsf{a}_3, \ldots)) \equiv \mathsf{f}\ \mathsf{a}_2$$
$$\mathsf{mapStr}_i\ \mathsf{f}\ (\overline{\mathsf{tail}}_i^{\mathsf{A}}\ (\mathsf{a}_2, \mathsf{a}_3, \ldots)) \equiv \mathsf{mapStr}_i\ \mathsf{f}\ (\mathsf{a}_3, \ldots)$$

and so on, i.e. the output of $\mathsf{mapStr}_i\ \mathsf{f}\ (\mathsf{a}_1, \mathsf{a}_2, \mathsf{a}_3, \ldots)$ will be the stream:

$$(\mathsf{f}\ \mathsf{a}_1,\ \mathsf{f}\ \mathsf{a}_2,\ \mathsf{f}\ \mathsf{a}_3,\ \ldots)$$

This example shows that we need to play a lot with universe levels. However, we quickly reach difficulties. For example, there is an *up cast* function $\uparrow\colon (\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_{i+1}\ \mathsf{A})$ defined by the corecursive equations (for any $\mathsf{w}\colon \mathsf{Stream}_i\ \mathsf{A}$):[60]

$$\overline{\mathsf{head}}_{i+1}\ (\uparrow \mathsf{w}) = \overline{\mathsf{head}}_i\ \mathsf{w}$$
$$\overline{\mathsf{tail}}_{i+1}\ (\uparrow \mathsf{w}) = \uparrow (\overline{\mathsf{tail}}_i\ \mathsf{w})$$

i.e. the up cast function behaves like an identity function, mapping each stream into a copy at a higher universe level.

Is there a *down cast* function $\downarrow\colon (\mathsf{Stream}_{i+1}\ \mathsf{A}) \to (\mathsf{Stream}_i\ \mathsf{A})$? Clearly, the down cast function $\downarrow$ cannot be defined using the coiteration principle, because its domain level is higher than its codomain level. Notice that this situation mirrors the one discussed in Section 3.2.4, where we were able to construct a down cast function $\downarrow$, but the up cast function $\uparrow$ was missing. Instead, in this section we have the up cast function $\uparrow$, but the down cast function $\downarrow$ is missing.

As discussed for natural numbers in Section 3.2.4, one possible approach for defining the down cast function is to use the Principle of unique choice (Lemma 1.11.8). However, the author has not explored this possibility.

The question of whether or not the down cast function exists is an interesting one, because its existence implies that $\mathsf{Stream}_i\ \mathsf{A}$ and $\mathsf{Stream}_{i+1}\ \mathsf{A}$ are equivalent types. If all $\mathsf{Stream}_i\ \mathsf{A}$ are mutually equivalent, we do not have to worry about universe levels when defining functions between them, like the mapStr_i function above.

One approach to prove the mutual equivalence of all $\mathsf{Stream}_i\ \mathsf{A}$, is to augment HoTT with inference rules for manipulating streams. We would have to add a formation rule for streams, "destruction" rules like: "If s is a stream, then $(\mathsf{head}\ \mathsf{s})$ is of type A and $(\mathsf{tail}\ \mathsf{s})$ is a stream", and rules mirroring the coinduction principle and the coiteration principle. We will not do this, as we want to remain within the theory fleshed out in Chapter 1. Also, Theorem 3.3.32 should be a warning for indiscriminately adding inference rules into HoTT. For example, if we add inference rules for the powertype functor, we will be implicitly adding a final coalgebra for the functor, contradicting Theorem 3.3.32.

Instead, the approach we will follow consists on finding a primitive HoTT type T, so that each $\mathsf{Stream}_i\ \mathsf{A}$ is equivalent to T. If streams are like infinite lists, then they can be seen as functions with domain the natural numbers, i.e. elements of type $\mathbb{N} \to \mathsf{A}$. Hence, $\mathsf{Stream}_i\ \mathsf{A}$ and $\mathbb{N} \to \mathsf{A}$ should be equivalent types. This is what we prove next.

Theorem 3.3.41. *Let $i > 0$ be a universe level and $\mathsf{A}\colon \mathbf{Set}_{i-1}$ a set (hence, A is also at level i). Then, type $\mathsf{Stream}_i\ \mathsf{A}$ is equivalent to type $\mathbb{N} \to \mathsf{A}$. In other words, the following type is inhabited:*

$$(\mathsf{Stream}_i\ \mathsf{A}) \simeq (\mathbb{N} \to \mathsf{A})$$

Proof. First, we define an auxiliary function by recursion on $\mathbb{N}$ (Lemma 1.5.53). This function composes a given function $\mathsf{f}\colon \mathsf{C} \to \mathsf{C}$ an specified number of times n. So, given $\mathsf{C}\colon \mathcal{U}_j$ and $\mathsf{f}\colon \mathsf{C} \to \mathsf{C}$, define $\mathsf{iter}\ \mathsf{f}\colon \mathbb{N} \to \mathsf{C} \to \mathsf{C}$ by the equations (for any $\mathsf{n}\colon \mathbb{N}$):

$$\mathsf{iter}\ \mathsf{f}\ 0 :\equiv \mathsf{id}_{\mathsf{C}}$$
$$\mathsf{iter}\ \mathsf{f}\ (\mathsf{succ}\ \mathsf{n}) :\equiv (\mathsf{iter}\ \mathsf{f}\ \mathsf{n}) \circ \mathsf{f}$$

[60] Notice that A is playing two roles. In $(\mathsf{Stream}_i\ \mathsf{A})$, type A is at level i, while in $(\mathsf{Stream}_{i+1}\ \mathsf{A})$, type A is at level $i+1$. This is allowed by the cumulative universe rule, i.e. if $\mathsf{A}\colon \mathcal{U}_i$, then also $\mathsf{A}\colon \mathcal{U}_{i+1}$.

Now, define $\mathsf{f}_1 : (\mathsf{Stream}_i\ \mathsf{A}) \to (\mathbb{N} \to \mathsf{A})$ as:

$$\mathsf{f}_1\ \mathsf{s} :\equiv \lambda \mathsf{n} : \mathbb{N}.\ \overline{\mathsf{head}}_i\ (\mathsf{iter}\ \overline{\mathsf{tail}}_i\ \mathsf{n}\ \mathsf{s})$$

We will define function $\mathsf{f}_2 : (\mathbb{N} \to \mathsf{A}) \to (\mathsf{Stream}_i\ \mathsf{A})$ by the coiteration principle for $\mathsf{Stream}_i\ \mathsf{A}$. The principle can be used because A is at level $i-1$, which implies that $\mathbb{N} \to \mathsf{A}$ is at level $i-1$ (type $\mathbb{N}$ is at every universe level, see its formation rule on page 42).

So, define f_2 by the equations (for any $\mathsf{g} : \mathbb{N} \to \mathsf{A}$):[61]

$$\begin{aligned} \overline{\mathsf{head}}_i\ (\mathsf{f}_2\ \mathsf{g}) &= \mathsf{g}\ 0 \\ \overline{\mathsf{tail}}_i\ (\mathsf{f}_2\ \mathsf{g}) &= \mathsf{f}_2\ (\lambda \mathsf{n} : \mathbb{N}.\ \mathsf{g}\ (\mathsf{succ}\ \mathsf{n})) \end{aligned}$$

In other words, given g, function f_2 will produce the stream $(\mathsf{g}\ 0,\ \mathsf{g}\ 1,\ \mathsf{g}\ 2,\ \ldots)$.

Next, we prove that f_1 and f_2 are inverses of each other.

Claim 1: Type $\prod_{\mathsf{g}:\mathbb{N}\to\mathsf{A}} \mathsf{f}_1\ (\mathsf{f}_2\ \mathsf{g}) = \mathsf{g}$ is inhabited.

Let $\mathsf{g} : \mathbb{N} \to \mathsf{A}$. We are trying to prove that two functions are identifiable. Therefore, by function extensionality, it is enough to prove:

$$\prod_{\mathsf{n}:\mathbb{N}} \mathsf{f}_1\ (\mathsf{f}_2\ \mathsf{g})\ \mathsf{n} = \mathsf{g}\ \mathsf{n} \tag{3.38}$$

However, if we try to prove (3.38) by induction on natural numbers, we will get stuck in the inductive step, because (3.38) does not produce an inductive hypothesis that is sufficiently general. So, we will prove the following type instead:

$$\prod_{\mathsf{n}:\mathbb{N}} \prod_{\mathsf{t}:\mathbb{N}\to\mathsf{A}} \mathsf{f}_1\ (\mathsf{f}_2\ \mathsf{t})\ \mathsf{n} = \mathsf{t}\ \mathsf{n} \tag{3.39}$$

If type (3.39) is inhabited, then it trivially follows that type (3.38) is inhabited, i.e. just instantiate (3.39) with function g.

We prove (3.39) by induction on $\mathbb{N}$ (Lemma 1.5.52). So, we need to prove that the following types are inhabited:

$$\prod_{\mathsf{t}:\mathbb{N}\to\mathsf{A}} \mathsf{f}_1\ (\mathsf{f}_2\ \mathsf{t})\ 0 = \mathsf{t}\ 0$$

$$\prod_{\mathsf{n}:\mathbb{N}} \left(\prod_{\mathsf{t}:\mathbb{N}\to\mathsf{A}} \mathsf{f}_1\ (\mathsf{f}_2\ \mathsf{t})\ \mathsf{n} = \mathsf{t}\ \mathsf{n} \right) \to \left(\prod_{\mathsf{r}:\mathbb{N}\to\mathsf{A}} \mathsf{f}_1\ (\mathsf{f}_2\ \mathsf{r})\ (\mathsf{succ}\ \mathsf{n}) = \mathsf{r}\ (\mathsf{succ}\ \mathsf{n}) \right)$$

The first type is inhabited as follows:

$$\begin{aligned} \mathsf{f}_1\ (\mathsf{f}_2\ \mathsf{t})\ 0 &\overset{(1)}{=} \overline{\mathsf{head}}_i\ (\mathsf{iter}\ \overline{\mathsf{tail}}_i\ 0\ (\mathsf{f}_2\ \mathsf{t})) \\ &\overset{(2)}{=} \overline{\mathsf{head}}_i\ (\mathsf{f}_2\ \mathsf{t}) \\ &\overset{(3)}{=} \mathsf{t}\ 0 \end{aligned}$$

where (1) follows by definition of f_1, (2) by definition of iter applied on 0, and (3) by definition of f_2 on its $\overline{\mathsf{head}}_i$ case.

For the second type, let $\mathsf{n} : \mathbb{N}$ with inductive hypothesis $\prod_{\mathsf{t}:\mathbb{N}\to\mathsf{A}} \mathsf{f}_1\ (\mathsf{f}_2\ \mathsf{t})\ \mathsf{n} = \mathsf{t}\ \mathsf{n}$. Then, we have:

$$\begin{aligned} \mathsf{f}_1\ (\mathsf{f}_2\ \mathsf{r})\ (\mathsf{succ}\ \mathsf{n}) &\overset{(1)}{=} \overline{\mathsf{head}}_i\ (\mathsf{iter}\ \overline{\mathsf{tail}}_i\ (\mathsf{succ}\ \mathsf{n})\ (\mathsf{f}_2\ \mathsf{r})) \\ &\overset{(2)}{=} \overline{\mathsf{head}}_i\ (\mathsf{iter}\ \overline{\mathsf{tail}}_i\ \mathsf{n}\ (\overline{\mathsf{tail}}_i\ (\mathsf{f}_2\ \mathsf{r}))) \end{aligned}$$

[61] To conform to the coiteration principle, the first equation is using function $\mathsf{r} :\equiv \lambda \mathsf{g} : \mathbb{N} \to \mathsf{A}.\ \mathsf{g}\ 0$, so that $\overline{\mathsf{head}}_i\ (\mathsf{f}_2\ \mathsf{g}) = \mathsf{r}\ \mathsf{g}$ is definitionally equal to $\overline{\mathsf{head}}_i\ (\mathsf{f}_2\ \mathsf{g}) = \mathsf{g}\ 0$.

The second equation is using function $\mathsf{t} :\equiv \lambda \mathsf{g} : \mathbb{N} \to \mathsf{A}.\ \lambda \mathsf{n} : \mathbb{N}.\ \mathsf{g}\ (\mathsf{succ}\ \mathsf{n})$, so that $\overline{\mathsf{tail}}_i\ (\mathsf{f}_2\ \mathsf{g}) = \mathsf{f}_2\ (\mathsf{t}\ \mathsf{g})$ is definitionally equal to $\overline{\mathsf{tail}}_i\ (\mathsf{f}_2\ \mathsf{g}) = \mathsf{f}_2\ (\lambda \mathsf{n} : \mathbb{N}.\ \mathsf{g}\ (\mathsf{succ}\ \mathsf{n}))$.

$$\overset{(3)}{=} \mathsf{f}_1\ (\overline{\mathsf{tail}}_i\ (\mathsf{f}_2\ \mathsf{r}))\ \mathsf{n}$$
$$\overset{(4)}{=} \mathsf{f}_1\ (\mathsf{f}_2\ (\lambda \mathsf{m}:\mathbb{N}.\ \mathsf{r}\ (\mathsf{succ}\ \mathsf{m})))\ \mathsf{n}$$
$$\overset{(5)}{=} (\lambda \mathsf{m}:\mathbb{N}.\ \mathsf{r}\ (\mathsf{succ}\ \mathsf{m}))\ \mathsf{n}$$
$$\overset{(6)}{=} \mathsf{r}\ (\mathsf{succ}\ \mathsf{n})$$

where (1) follows by definition of f_1, (2) by definition of iter applied on a successor, (3) by definition of f_1, (4) by definition of f_2 on its $\overline{\mathsf{tail}}_i$ case, (5) by the inductive hypothesis, and (6) by computation.

This proves the claim.

Claim 2: Type $\prod_{\mathsf{s}:\mathsf{Stream}_i\ \mathsf{A}} \mathsf{f}_2\ (\mathsf{f}_1\ \mathsf{s}) = \mathsf{s}$ is inhabited.

Let $\mathsf{s}:\mathsf{Stream}_i\ \mathsf{A}$. We will use the coinduction principle for $\mathsf{Stream}_i\ \mathsf{A}$ (Theorem 3.3.39) to conclude that $\mathsf{f}_2\ (\mathsf{f}_1\ \mathsf{s}) = \mathsf{s}$ is inhabited.

Define:

$$\mathsf{R}\ \mathsf{a}\ \mathsf{b} :\equiv \sum_{\mathsf{w}:\mathsf{Stream}_i\ \mathsf{A}} (\mathsf{a} = \mathsf{f}_2\ (\mathsf{f}_1\ \mathsf{w})) \times (\mathsf{b} = \mathsf{w})$$

having type $(\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_i\ \mathsf{A}) \to \mathcal{U}_i$ (see Remark 1.5.29 and formation rule for identity types on page 45).

We have that $\mathsf{R}\ (\mathsf{f}_2\ (\mathsf{f}_1\ \mathsf{s}))\ \mathsf{s}$ is inhabited by term:

$$(\ \mathsf{s},\ (\mathsf{refl}_{(\mathsf{Stream}_i\ \mathsf{A})}\ (\mathsf{f}_2\ (\mathsf{f}_1\ \mathsf{s})),\ \mathsf{refl}_{(\mathsf{Stream}_i\ \mathsf{A})}\ \mathsf{s})\ )$$

So, it remains to show that R is a stream bisimulation, i.e. the following type is inhabited:

$$\prod_{\mathsf{m},\mathsf{n}:\mathsf{Stream}_i\ \mathsf{A}} (\mathsf{R}\ \mathsf{m}\ \mathsf{n}) \to \left((\overline{\mathsf{head}}_i\ \mathsf{m} = \overline{\mathsf{head}}_i\ \mathsf{n}) \times (\mathsf{R}\ (\overline{\mathsf{tail}}_i\ \mathsf{m})\ (\overline{\mathsf{tail}}_i\ \mathsf{n}))\right)$$

Let $\mathsf{m}, \mathsf{n}:\mathsf{Stream}_i\ \mathsf{A}$ with $\mathsf{h}_1:\mathsf{R}\ \mathsf{m}\ \mathsf{n}$. By definition of $\mathsf{R}\ \mathsf{m}\ \mathsf{n}$, we have $\mathsf{w}:\mathsf{Stream}_i\ \mathsf{A}$ with the following inhabited types:

$$\mathsf{m} = \mathsf{f}_2\ (\mathsf{f}_1\ \mathsf{w}) \qquad \mathsf{n} = \mathsf{w} \tag{3.40}$$

Then, we have:

$$\overline{\mathsf{head}}_i\ \mathsf{m} \overset{(1)}{=} \overline{\mathsf{head}}_i\ (\mathsf{f}_2\ (\mathsf{f}_1\ \mathsf{w}))$$
$$\overset{(2)}{=} \mathsf{f}_1\ \mathsf{w}\ 0$$
$$\overset{(3)}{=} \overline{\mathsf{head}}_i\ (\mathsf{iter}\ \overline{\mathsf{tail}}_i\ 0\ \mathsf{w})$$
$$\overset{(4)}{=} \overline{\mathsf{head}}_i\ \mathsf{w}$$
$$\overset{(5)}{=} \overline{\mathsf{head}}_i\ \mathsf{n}$$

where (1) holds because $\mathsf{m} = \mathsf{f}_2\ (\mathsf{f}_1\ \mathsf{w})$ is inhabited by Equations (3.40), (2) follows by definition of f_2 on its $\overline{\mathsf{head}}_i$ case, (3) by definition of f_1, (4) by definition of iter applied on 0, and (5) holds because $\mathsf{n} = \mathsf{w}$ is inhabited by Equations (3.40).

We also have:

$$\overline{\mathsf{tail}}_i\ \mathsf{m} \overset{(1)}{=} \overline{\mathsf{tail}}_i\ (\mathsf{f}_2\ (\mathsf{f}_1\ \mathsf{w}))$$
$$\overset{(2)}{=} \mathsf{f}_2\ (\lambda \mathsf{r}:\mathbb{N}.\ \mathsf{f}_1\ \mathsf{w}\ (\mathsf{succ}\ \mathsf{r}))$$
$$\overset{(3)}{=} \mathsf{f}_2\ (\lambda \mathsf{r}:\mathbb{N}.\ \overline{\mathsf{head}}_i\ (\mathsf{iter}\ \overline{\mathsf{tail}}_i\ (\mathsf{succ}\ \mathsf{r})\ \mathsf{w}))$$
$$\overset{(4)}{=} \mathsf{f}_2\ (\lambda \mathsf{r}:\mathbb{N}.\ \overline{\mathsf{head}}_i\ (\mathsf{iter}\ \overline{\mathsf{tail}}_i\ \mathsf{r}\ (\overline{\mathsf{tail}}_i\ \mathsf{w})))$$
$$\overset{(5)}{=} \mathsf{f}_2\ (\lambda \mathsf{r}:\mathbb{N}.\ \mathsf{f}_1\ (\overline{\mathsf{tail}}_i\ \mathsf{w})\ \mathsf{r})$$

$$\overset{(6)}{=} \mathsf{f}_2\ (\mathsf{f}_1\ (\overline{\mathsf{tail}}_i\ \mathsf{w}))$$
$$\overset{(7)}{=} \mathsf{f}_2\ (\mathsf{f}_1\ (\overline{\mathsf{tail}}_i\ \mathsf{n}))$$

where (1) holds because $\mathsf{m} = \mathsf{f}_2\ (\mathsf{f}_1\ \mathsf{w})$ is inhabited by Equations (3.40), (2) follows by definition of f_2 on its $\overline{\mathsf{tail}}_i$ case, (3) by definition of f_1, (4) by definition of iter applied on a successor, (5) by definition of f_1, (6) by the uniqueness of Π-types rule on page 21, and (7) holds because $\mathsf{n} = \mathsf{w}$ is inhabited by Equations (3.40).

Therefore, we have a proof $\mathsf{p}_1 : \overline{\mathsf{tail}}_i\ \mathsf{m} = \mathsf{f}_2\ (\mathsf{f}_1\ (\overline{\mathsf{tail}}_i\ \mathsf{n}))$, which implies that type $\mathsf{R}\ (\overline{\mathsf{tail}}_i\ \mathsf{m})\ (\overline{\mathsf{tail}}_i\ \mathsf{n})$ is inhabited by term:

$$(\ \overline{\mathsf{tail}}_i\ \mathsf{n},\ (\mathsf{p}_1,\ \mathsf{refl}_{(\mathsf{Stream}_i\ \mathsf{A})}\ (\overline{\mathsf{tail}}_i\ \mathsf{n}))\ )$$

Hence, R is a stream bisimulation, and this proves the claim.

□

Therefore, if we take some set $\mathsf{A} : \mathbf{Set}_0$, we will have that all (for $i > 0$) $\mathsf{Stream}_i\ \mathsf{A}$ are mutually equivalent. From here, using univalence, one can prove that $\mathsf{Stream}_i\ \mathsf{A}$ *can* be given the structure of a final $\mathsf{StreamF}_i\ \mathsf{A}$-coalgebra (and not just merely a *lower level* final coalgebra). But we omit the details, as this is not critical for the example.

Recapitulating, we showed that all types $\mathsf{Stream}_i\ \mathsf{A}$ are mutually equivalent by proving that each $\mathsf{Stream}_i\ \mathsf{A}$ is equivalent to the function type $\mathbb{N} \to \mathsf{A}$.

However, we should not dismiss investigating the construction of a down cast function ↓ (see discussion before Theorem 3.3.41), because its existence *seems* to characterize final coalgebras, i.e. the up cast function ↑ is an equivalence (having as inverse the down cast function ↓) *if and only if* $\mathsf{Stream}_i\ \mathsf{A}$ can be given the structure of a final $\mathsf{StreamF}_i\ \mathsf{A}$-coalgebra. Further exploration is required on this.

3.3.5 Discussion

In this section, we discuss particularities and limitations of the approach leading to Corollary 3.3.31.

The first limitation is that our functors need to preserve sets. This has as consequence that all functor parameters have to be sets (for example, parameter A in functor $\mathsf{StreamF}_i\ \mathsf{A}$ has to be a set). In what follows, we explain why we had to impose this limitation to the results of Section 3.3.2.

In the proof of Lemma 3.3.25, we defined some predicate P as:

$$\lambda \mathsf{w} : \mathsf{B}.\ \sum_{\mathsf{y}:\mathsf{A}_\mathsf{F}} \left\| \sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}] = \mathsf{y}) \times (\mathsf{h}\ \mathsf{c} = \mathsf{w}) \right\| \tag{3.41}$$

which expresses how some required function should behave. Later in the proof, this definition of P leads to a proof that the following type is inhabited:

$$\prod_{\mathsf{y}_1,\mathsf{y}_2:\mathsf{A}_\mathsf{F}} \left\| \sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}] = \mathsf{y}_1) \times (\mathsf{h}\ \mathsf{c} = \mathsf{w}) \right\| \to \left\| \sum_{\mathsf{c}:\mathsf{A}} ([\mathsf{c}] = \mathsf{y}_2) \times (\mathsf{h}\ \mathsf{c} = \mathsf{w}) \right\| \to (\mathsf{y}_1 = \mathsf{y}_2)$$

If $\mathsf{y}_1 = \mathsf{y}_2$ is not a mere proposition, then we *cannot* use (-1)-truncation recursion to remove the (-1)-truncations on the hypotheses. Therefore, this leads to the restriction:

- Type A_F must be a set.

Type A_F was defined as the quotient over some behavior relation. Quotients were defined as 0-truncations of some coequalizer (see proof of Lemma 1.11.24). However, in Section 1.11.5 it was not explained *why* quotients need to be sets. Since A_F must be a set, using 0-truncations in the definition of quotients will ensure that this restriction on A_F is satisfied.

Since quotients use 0-truncations, their recursion principle can be used only for constructing functions whose codomain is a set (see Corollary 1.11.14 and Lemma 1.11.26). In the proof of Lemma 3.3.26, we needed to construct a function of type $\mathsf{A}_\mathsf{F} \to (\mathsf{H}\ \mathsf{A}_\mathsf{F})$. The natural thing to do here is to use the recursion principle for quotients. However, $\mathsf{H}\ \mathsf{A}_\mathsf{F}$ *must* be a set if we want to use the recursion principle. The easiest way to achieve this is to require that functor H preserves sets (so that $\mathsf{H}\ \mathsf{A}_\mathsf{F}$ will be a set).

Even though $\mathsf{A_F}$ is required to be a set, we did not have to specialize Definitions 1.10.4 and 1.10.8 to sets, as we had to do in Section 3.2 for their algebra counterparts. The reason for this is that on the critical lemmas of Section 3.3.2, we never work at the level of an abstract coalgebra, but we always use the quotient $\mathsf{A_F}$. Another reason for not specializing Definitions 1.10.4 and 1.10.8 to sets is that there is no restriction on the universally quantified coalgebra B in Definition 1.10.8.

Therefore, if we want to remove the "set preservation property" on our functor, we need to find a way to remove the (-1)-truncation in the definition of predicate P (function (3.41) above), as this will remove the restriction that $\mathsf{A_F}$ must be a set.

The second obvious limitation is that our experiment only produces lower level final coalgebras, which are weaker than final coalgebras. Nevertheless, lower level final coalgebras are still useful concepts, as they are capable of defining *unique* functions by corecursive equations.

3.4 Chapter summary

This chapter carried out the generalization of the Knaster-Tarski Lemma into the language of algebras and coalgebras. The intention on doing such generalization was to explore the possibility of constructing initial algebras (i.e. inductive types) and final coalgebras (i.e. coinductive types). The generalization followed some of the analogies between order theory and category theory.

Although the experiment fails to produce initial/final (co)algebras, it produces structures that are capable of defining unique functions by (co)recursion. These structures were baptized lower level initial/final (co)algebras, because they behave like an initial/final (co)algebra but only relative to (co)algebras at a strictly lower universe level.

Section 3.1 gave an overview of the most common approach used in the literature to compute initial/final (co)algebras. The approach involves the computation of colimits of ω-chains and limits of ω-cochains. The approach also served to explain the analogy between order theory and category theory.

Sections 3.2 and 3.3 developed the suggested generalization of Knaster-Tarski for inductive and coinductive types, respectively. First, both sections showed how the generalization produces lower level *weakly* initial/final (co)algebras (Subsections 3.2.1 and 3.3.1, respectively). Then, it was shown how these lower level weakly initial/final (co)algebras can be refined into lower level initial/final (co)algebras (Subsections 3.2.2 and 3.3.2, respectively). An example of endofunctor without an initial/final (co)algebra, but having a lower level initial/final (co)algebra, was presented in Subsections 3.2.3 and 3.3.3, respectively. Two examples were developed in Subsections 3.2.4 and 3.3.4 showing how lower level initial/final (co)algebras are used. Finally, limitations on the approach were discussed, together with explanations of some decisions made by the author (Subsections 3.2.5 and 3.3.5, respectively).

Chapter 4

Conclusions

In order to explore the possibility of building initial algebras (i.e. inductive types) and final coalgebras (i.e. coinductive types) in HoTT, this report investigated the generalization of the Knaster-Tarski Lemma[1] into the language of algebras and coalgebras. The generalization followed some of the analogies between order theory and category theory (see Section 3.1).

The particular approach followed on this report for doing the aforementioned generalization produced the following consequences:

- For inductive types, there are no required conditions on the endofunctor (see Theorem 3.2.21 and Corollary 3.2.22).

  For coinductive types, the only required condition on the endofunctor is that it needs to preserve sets (see Theorem 3.3.30 and Corollary 3.3.31).

  In contrast, common iterative approaches require that endofunctors preserve colimits of chains and limits of cochains (see Section 3.1).

- For inductive types, the constructed structures fail to be initial algebras only by a universe level. We called these structures *lower level initial set algebras* (see Definition 3.2.9 and Corollary 3.2.22), because they behave like initial algebras relative to *set* algebras that are at a *strictly lower* universe level than the universe level where the lower level initial set algebra lives in.

  For coinductive types, the constructed structures fail to be final coalgebras only by a universe level. We called these structures *lower level final coalgebras* (see Definition 3.3.8 and Corollary 3.3.31), because they behave like final coalgebras relative to coalgebras that are at a *strictly lower* universe level than the universe level where the lower level final coalgebra lives in.

  Notice that lower level final coalgebras are defined relative to *types* at a lower level, while lower level initial set algebras are defined relative to *sets* at a lower level.

- In spite of this restriction on universe levels, it was shown that lower level initial set algebras are capable of defining *unique* functions by recursive equations, as initial algebras are able to. This ability was demonstrated with a detailed example in Section 3.2.4, where the usual iteration and induction principles for the natural numbers (seen as a lower level initial set algebra) were derived.

  Similarly, it was shown that lower level final coalgebras are capable of defining *unique* functions by corecursive equations, as final coalgebras are able to. This ability was demonstrated with a detailed example in Section 3.3.4, where the usual coiteration and coinduction principles for streams (seen as a lower level final coalgebra) were derived.

  However, as the examples in Sections 3.2.4 and 3.3.4 showed, we need to play a lot with universe levels when defining functions using the (co)iteration principles, due to the intrinsic restriction on universe levels forced on us by lower level initial/final (co)algebras. This is a clear disadvantage.

[1] Lemma 2.3.6.

- Theorems 3.2.21 and 3.3.30, which are the last steps of the "refinement processes" in Sections 3.2.2 and 3.3.2, can be summarized in the following table:[2]

If we have...	***... then we can construct***
Lower level weakly initial set algebra	Lower level initial set algebra
Weakly initial set algebra	Initial set algebra
Lower level weakly final coalgebra	Lower level final coalgebra
Weakly final coalgebra	Final coalgebra

  In particular, these processes showed that the refinement of a lower level weakly initial set algebra into a lower level initial set algebra *uses the same steps and proofs* as the refinement of a weakly initial set algebra into an initial set algebra.

  Similarly, the refinement of a lower level weakly final coalgebra into a lower level final coalgebra *uses the same steps and proofs* as the refinement of a weakly final coalgebra into a final coalgebra.

- Lower level initial/final (co)algebras are weaker concepts than initial/final (co)algebras. Evidence for this is the fact that any set-preserving endofunctor has a lower level initial/final (co)algebra (Corollaries 3.2.22 and 3.3.31), but not every set-preserving endofunctor has an initial/final (co)algebra. An example of such functor is the powertype functor,[3] as shown by Theorems 3.2.25 and 3.3.32.

  Another piece of evidence is that Lambek's Lemma[4] does not hold in general for lower level initial/final (co)algebras, as shown by Corollaries 3.2.27 and 3.3.34.

- A big limitation of the approach presented on this report is that it can only build sets, not general types. Sections 3.2.5 and 3.3.5 explained the origins of this limitation.

This report also showed the following:

- Chapter 2 developed an example of set-based mathematics formalized in HoTT. More specifically, it formalized basic order-theoretic concepts, like poset, monotone function, complete lattice, infima, suprema, meet, join, fixpoint, greatest and least fixpoint, the Knaster-Tarski Lemma, and (co)induction principles.

- Chapter 2 made use of the "ignore universe levels" style of doing mathematics in HoTT. The chapter also highlighted an edge case for this style at Example 2.2.8, where an impredicative definition was made.

- In contrast, Chapter 3 made use of the "explicit universe levels" style of doing mathematics in HoTT.

  The chapter justified the use of this style by explaining that lower level initial/final (co)algebras are different from initial/final (co)algebras only by a quantification at different universe levels (see Definitions 3.2.9 and 3.3.8). Therefore, if universe levels were ignored, we would get the *apparent* contradiction that any set-preserving endofunctor has an initial/final (co)algebra, while at the same time, the powertype functor does not have one!

Before ending this report, we provide a list of possible questions to investigate for future work. Most of these questions are a restatement of the limitations described at Sections 3.2.5 and 3.3.5.

- Can dependence on the results of Chapter 2 be removed from the refinement process on Section 3.2.2? Will this be enough for the refinement process to be generalized to arbitrary types?

- Can Lemma 3.3.25 be proved without using (-1)-truncations on property (3.41), as explained in Section 3.3.5? Will this be enough for the refinement process to be generalized to quotients that are not sets?

- In Section 3.2.5 it was explained that the "morphism 0-truncation property" (Definition 3.2.35) *seems* to be required in order to obtain a (lower level) weakly initial *set* algebra by 0-truncating a (lower level) weakly initial algebra. Is the "morphism 0-truncation property" a necessary condition?

[2] See Definitions 3.2.5, 3.3.4, 3.2.9, and 3.3.8.
[3] See Definition 3.2.23 and Lemma 3.2.24.
[4] See Lemmas 3.2.11 and 3.3.10.

- At step 1 of the refinement process in Section 3.3.2, it was stated that bisimilarity and behavioral equivalence are logically equivalent when functor H preserves weak pullbacks.[5]

  We chose behavioral equivalence for the refinement process because it is always an equivalence relation, while bisimilarity may fail to satisfy transitivity. However, when H preserves weak pullbacks, bisimilarity *is* an equivalence relation, because it coincides with behavioral equivalence on this case.

  Can we construct a final coalgebra by using bisimilarity, without assuming that H preserves weak pullbacks?

  Notice that we do *not* need to prove that a relation is an equivalence relation to be able to define a quotient (see Lemma 1.11.24).

- Sections 3.2.4 and 3.3.4 raised a very interesting problem.

  In Section 3.2.4, it was explained that we need to carefully manipulate universe levels in order to define functions between types Nat_i (i.e. the lower level initial set algebra for the natural numbers functor NatF_i).

  If we are able to prove that all Nat_i are mutually equivalent (for $i > 0$), then there is no need to worry about universe levels when defining functions.

  Section 3.2.4 proved the mutual equivalence between all Nat_i, by showing that each one of them is equivalent to type $\mathbb{N}$ (see Theorem 3.2.34).

  But Section 3.2.4 suggested another possibility for showing mutual equivalence: if we are able to prove that there is an up cast function $\uparrow\colon \mathsf{Nat}_i \to \mathsf{Nat}_{i+1}$ acting as inverse to the (always existing) down cast function $\downarrow\colon \mathsf{Nat}_{i+1} \to \mathsf{Nat}_i$, then all Nat_i are mutually equivalent.

  Also, by using the fact that Nat_i is equivalent to $\mathbb{N}$ (Theorem 3.2.34), it *seems* that the following statement is provable (however, the author has not verified this).

  > For every $i > 0$, Nat_i can be given the structure of an initial set algebra *if and only if* for every $i > 0$, the down cast function $\downarrow\colon \mathsf{Nat}_{i+1} \to \mathsf{Nat}_i$ is an equivalence.

  Can we prove this statement without using Theorem 3.2.34?

  The pattern is similar for the streams example in Section 3.3.4, but the roles for the up and down cast functions are reversed: if we are able to prove that there is a down cast function $\downarrow\colon (\mathsf{Stream}_{i+1}\ \mathsf{A}) \to (\mathsf{Stream}_i\ \mathsf{A})$ acting as inverse to the (always existing) up cast function $\uparrow\colon (\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_{i+1}\ \mathsf{A})$, then all $(\mathsf{Stream}_i\ \mathsf{A})$ are mutually equivalent.

  And again, by using the fact that $\mathsf{Stream}_i\ \mathsf{A}$ is equivalent to $\mathbb{N} \to \mathsf{A}$ (Theorem 3.3.41), it *seems* that the following statement is provable (again, the author has not verified this).

  > For every $i > 0$, $\mathsf{Stream}_i\ \mathsf{A}$ can be given the structure of a final coalgebra *if and only if* for every $i > 0$, the up cast function $\uparrow\colon (\mathsf{Stream}_i\ \mathsf{A}) \to (\mathsf{Stream}_{i+1}\ \mathsf{A})$ is an equivalence.

  Can we prove this statement without using Theorem 3.3.41?

  The two boxed statements suggest that there might be a general pattern. Is it possible to generalize these statements to arbitrary lower level initial/final (co)algebras?

  Can initial/final (co)algebras be characterized by lower level initial/final (co)algebras through the use of the up and down cast functions, as in the boxed statements above? Is this question even sensible? Further investigation is required.

[5] See Definitions 3.3.15, 3.3.19, and 3.3.20. See also Definition 175 and Theorem 177 in Chapter 6 of [7].

References

[1] Jiří Adámek, Stefan Milius, and Lawrence S. Moss. Initial algebras and terminal coalgebras: a survey. Draft as of July 7, 2010. Available at: https://www.tu-braunschweig.de/Medien-DB/iti/survey_full.pdf.

[2] Marcelo Aguilar, Samuel Gitler, and Carlos Prieto. *Algebraic Topology from a Homotopical Viewpoint*. Universitext. Springer-Verlag, 2002.

[3] Ki Yung Ahn. *The Nax Language: Unifying Functional Programming and Logical Reasoning in a Language based on Mendler-style Recursion Schemes and Term-indexed Types*. PhD thesis, Portland State University, 2014. Available at: http://pdxscholar.library.pdx.edu/open_access_etds/2088/.

[4] Benedikt Ahrens, Paolo Capriotti, and Régis Spadotti. Non-wellfounded trees in homotopy type theory. In *13th International Conference on Typed Lambda Calculi and Applications*, volume 38 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 17–30. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2015. arXiv:1504.02949.

[5] Steve Awodey. *Category Theory*, volume 52 of *Oxford Logic Guides*. Oxford University Press, 2nd edition, 2010.

[6] Steve Awodey and Michael A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, 2009. arXiv:0709.0248.

[7] Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors. *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*. Elsevier Science Inc., 2007.

[8] R. Bos and C. Hemerik. *An Introduction to the Category-theoretic Solution of Recursive Domain Equations*. Department of Mathematics and Computing Science. Eindhoven University of Technology, 1988.

[9] G. Gierz, K. H. Hofmann, K. Keimel, J. D. Lawson, M. Mislove, and D. S. Scott. *Continuous Lattices and Domains*, volume 93 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2003.

[10] Paul G. Goerss and John F. Jardine. *Simplicial Homotopy Theory*, volume 174 of *Progress in mathematics*. Birkhäuser Verlag, 1999.

[11] Ichiro Hasuo. Modal logics for coalgebras. Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2003. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.2003.

[12] Allen Hatcher. *Algebraic Topology*. Cambridge University Press, 2002.

[13] Fairouz Kamareddine, Twan Laan, and Rob Nederpelt. *A Modern Perspective on Type Theory: From its Origins until Today*, volume 29 of *Applied Logic Series*. Springer Science & Business Media, 2005.

[14] Chris Kapulkin, Peter LeFanu Lumsdaine, and Vladimir Voevodsky. The simplicial model of univalent foundations. 2012. arXiv:1211.2851.

[15] Per Martin-Löf. An intuitionistic theory of types. Department of Mathematics. University of Stockholm, 1972. Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.131.926.

[16] Davide Sangiorgi. *Introduction to Bisimulation and Coinduction*. Cambridge University Press, 2012.

[17] Davide Sangiorgi and Jan Rutten. *Advanced Topics in Bisimulation and Coinduction*, volume 52 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 2012.

[18] Morten Heine Sørensen and Paweł Urzyczyn. *Lectures on the Curry-Howard Isomorphism*, volume 149 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science Inc., 2006.

[19] Sam Staton. Relating coalgebraic notions of bisimulation. *Logical Methods in Computer Science*, 7(1:13), 2011. arXiv:1101.4223.

[20] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. https://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

[21] Benno van den Berg and Richard Garner. Topological and simplicial models of identity types. *ACM Transactions on Computational Logic*, 13(1:3), 2012. arXiv:1007.4638.

[22] *What is Formal Math?* vdash.org article available at: http://vdash.org/formal.

[23] Michael Alton Warren. *Homotopy Theoretic Aspects of Constructive Type Theory*. PhD thesis, Carnegie Mellon University, 2008. Available at: http://mawarren.net/papers/phd.pdf.

[24] The Coq Proof Assistant. Reference manual and software at: https://coq.inria.fr.

[25] The HoTT Coq Library. Source code and installation instructions at: https://github.com/HoTT/HoTT.

[26] Coq script for this document and instructions on how to run it are available at: https://github.com/jeshecdom/hott-knaster-tarski-experiment-dissertation.