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

HOMOTOPÍA DE TIPOS Y UNIVALENCIA

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

PRESENTA:
RICARDO MANSILLA SANCHEZ

TUTOR
DR. CARLOS PRIETO DE CASTRO
IMATE, UNAM

MÉXICO, D.F. OCTUBRE 2015

“If programming is understood not as the writing of instructions for this or that computing machine but as the design of methods of computation that it is the computer's duty to execute, then it no longer seems possible to distinguish the discipline of programming from constructive mathematics"

Per Martin-Löf

Índice general

Capítulo 1

Introducción

La Teoría de Tipos (TT) fue inventada por Bertrand Russel [1], pero fue con el trabajo de Alonzo Church [2] que se estableció como un sistema formal riguroso de uso teórico. La TT entonces comenzó a ser aplicada principalmente en el área del cómputo teórico de los lenguajes formales. Algunos años mas tarde, basado en el trabajo de Church, Per Martin-Löf [3] desarrolló una generalizacion de la teoría que es conocida como teoría de tipos intuicionista. Esta teoría ha sido reinterpretada recientemente a través de la teoría homotópica abstracta, sugiriendo un nuevo enfoque de los fundamentos matemáticos. Este enfoque brinda un contenido geométrico al carácter sintético de la teoría, preservando su implementabilidad computacional. Vladimir Voevodsky [4] propone un programa para reformular los fundamentos de la teoría matematica usando esta interpretación. Incluye además, en su programa, un axioma que contiene información novedosa en términos topológicos y lógicos, lo cual permite establecer un punto de partida para reinterpretar las matemáticas con un enfoque constructivo riguroso.

La TT se basa en *tipos*, que aunque son fundacionalmente distintos a los conjuntos, esta distinción es en ocasiones imprecisa. Por ejemplo un *tipo* ES un conjunto en el sentido de que define un objeto abstracto que codifica una colección de elementos. Sin embargo, el hecho de que dos elementos pueden estar representando lo mismo sin ser sintácticamente idénticos permite que dos elementos sean “iguales” sin ser “el mismo”. Finalmente, la infraestructura lógica permite definir el *tipo de conjuntos*, es decir, el tipo que contiene la información necesaria para decidir si algún otro tipo es en efectoi un conjunto o no. Citando la referencia principal de la tesis [4]

> *Intuitively, we would expect a type to "be a set" in this sense if it has no higher homotopical information: any two parallel paths are equal (up to homotopy), and similarly for parallel higher paths at all dimensions.*

de esta manera uno esperaría que los *tipos* se manifiesten como conjuntos *conjuntos* bajo determinadas condiciones. Tales condiciones están estrucutradas en la Teoría Homotópica de Tipos (HoTT).

En HoTT uno piensa en los *tipos* como espacios y las construcciones lógicas relacionadas con estos (producto interior y exterior, sumas, etc) como construcciones homotópicamente invariantes de estos espacios. De esta forma podemos manipular los espacios de forma "sintética" sin tener que definir topologías, geometrías o estructuras inherentes del espacio. Esta nueva interpretación de la homotopía piensa las igualdades $a = b$ de elementos $a, b : A$ como caminos $p : a \sim b$ del punto a al punto b en A. Esto también significa que dos funciones $f, g : A \to B$ son la misma solo en el caso de que exista una familia de caminos $p_x : f(x) \sim g(x)$ en B por cada $a : A$, es decir, que sean homotópicas.

En TT para cada tipo A existe un tipo Id_A, que codifica todas las identidades de elementos en A. En HoTT, esto se ve como la potencia A^I de los caminos posibles en A, es decir todos los $p : I \to A$. La existencia de este tipo identidad sugiere consecuencias que motivan lo que Voevodsky denominó *univalencia*. La univalencia esta codificada en un axioma conocido como *Axioma de Univalencia* que es nuevo y no forma parte de la TT clásica. La Univalencia permite hablar de "la igualdad entre tipos" como "el mismo tipo de homotopía", en otras palabras, podemos decir que existe un tipo $\mathcal{U}$ (usualmente conocido como *universo*) tal que sus elementos son tipos. Los tipos que son elementos de $\mathcal{U}$ son llamados tipos pequeños. Naturalmente esto nos permite decir que $\mathcal{U}$ no es pequeño y evitamos $\mathcal{U} : \mathcal{U}$. Como $\mathcal{U}$ es un tipo tenemos $Id_{\mathcal{U}}$, esto nos permite encontrar caminos $p : A \sim B$ en $\mathcal{U}$ que básicamente codifican la equivalencia homotópica $A \simeq B$. Lo anterior es lo mismo que decir que dados A y B hay una aplicación

$$Id_{\mathcal{U}}(A, B) \longrightarrow Eq(A, B)$$

del *tipo de identidad* $Id_{\mathcal{U}}$ al *tipo de equivalencia* de A y B. El Axioma de univalencia de Voevodsky establece la aplicación de regreso, permitiendo que se de la equivalencia de nivel superior

$$Id_{\mathcal{U}}(A, B) \simeq Eq(A, B)$$

En este trabajo pretendemos, de manera general dar una introducción formal a la teoría de tipos en el **Capítulo 1**. Definimos los tipos de funciones, así como las estructuras y patrones lógicos básicos usados para las construcciones de tipos. Terminamos el capítulo con una motivación de el carácter homotópico de dicha teoría.

En el **Capítulo 2** introducimos la homotopía de tipos y sus consecuencias. De manera que recorremos los conceptos establecidos en el **Capítulo 1** desde el punto de vista homotópico y demostramos algunas propiedades usando metodologías tanto de la teoría de tipos como de la teoría de categorías. El capítulo termina con el concepto de univalencia y promueve algun trabajo futuro sobre las consecuencias homotópicas en los grupoides de orden superior.

Finalmente en el **Capítulo 3** se estudia un ejemplo de univalencia en la estructura de semigrupos. También se describen brevemente cuales son los conflictos principales para exhibir univalencia en otras estructuras alebráicas, con lo que terminamos el trabajo.

Capítulo 2

Teoría de Tipos

2.1. Preliminares

En este trabajo, de manera muy general pretendemos dar una introducción formal a la teoría de tipos (Capítulo 1), introducir las nociones del carácter homotópico que tienen algunas de sus estructuras fundacionales (Capítulo 2) y a exhibir de manera explícita algunas de las propiedades de "orden superior" que podemos entontrar en las equivalencias de tipos (Capítulo 3).

La teoría de tipos es un lenguaje fundacional para reescribir la teoría matemática como alternativa del camino usual siguiendo la teoría de conjuntos basada en los axiomas de Zermelo-Fraenkel. La mayoría de los académicos reducen el significado del término al estudio de fomalismos abstractos como el cálculo lambda tipado pero su significado abarca todo lo que comprenda el estudio del diseño y análisis de los sistemas de tipos.

Existen algunos elementos muy particulares de la teoría de tipos pero la manera mas fácil de manifestar estas particularidades es comparar el esquema constructivo y relacional de los tipos con el de la teoría de conjuntos.

2.1.1. Tipos y conjuntos

La teoría de conjuntos no está basada en el estudio de "conjuntos". Es sobre todo el estudio de las interacciones de elementos abstractos definidos a partir de axiomas que existen dentro de la lógica de primer orden. Es decir, un sistema deductivo (como la

lógica de primer orden) permite definir un sistema axiomatico (como el de ZF). Que es el caso particular de como se define la teoría de conjuntos clásica. En cambio, en la teoría de tipos encontramos que no tenemos estas dos metaestructuras de objetos; en vez de tener conjuntos y proposiciones (operadores entre conjuntos que constituyen elementos de la logica proposicional), tenemos solo tipos. En la teoría de conjuntos la negación (operación complementaria) por ejemplo, constituye un elemento de la lógica de primer orden. Que a partir de los axiomas de ZF se convierte en un operador proposicional entre conjuntos. La teoría de tipos evita el problema al asignar un "tipo" para este tipo de operadores entre tipos, valga la redundancia. De esa forma una *proposición* es simplemente un tipo.

De esta forma podemos decir que los tipos, aunque no son conjuntos, son un poco como conjuntos en el sentido de que son de alguna forma contenedores de elementos. Es decir si existe la noción de pertenencia lo que sucede es que dicha noción esta manifestada en la sintaxis del conjunto, i.e los tipos dan información sintáctica, mientras los conjuntos dan información semántica. Por ejemplo, en la teoría de conjuntos escribimos

$$5 \in \{n \in \mathbb{N} \mid \forall x, y, z \in \mathbb{N}^{+}(x^n + y^n \neq z^n)\}$$

esta expresión *requiere* una prueba, es decir, uno debe probar que $(x^5 + y^5 \neq z^5)$, $\forall x, y, z \in \mathbb{N}$ para saber que 5 pertenece a este conjunto. En teoría de tipos uno escribe

$$4 + (8 * 3)^2 : \mathbb{N}$$

y esto dice que si la expresión anterior está **bien tipada** entonces $4 + (8 * 3)^2$, o el número que resulte de las reglas de computación y eliminación, es un natural. Esto lo veremos con mas detalle adelante, pero la idea básica es que sabiendo que $2, 3, 4$ y 8 son números naturales y que las operaciones $+, *$ y potencia son binarias que envían naturales en naturales, entonces podemos decir que ese número es natural. De manera que en la misma sintáxis de la expresión esta codificada la naturaleza del resultado y su tipo correspondiente.

Esta diferencia puede parecer trivial y tal vez no queda muy clara en un inicio, pero es la diferencia crucial entre la manera de hacer matemáticas con ambas teorías. Ya que es, por ejemplo, definitiva para tratar el problema de la *existencia*. El axioma del infinito nos asegura la existencia de conjuntos infinitos, y el axioma del conjunto potencia nos dice que el conjunto de los subconjuntos de cualquier conjunto existe. Esto le da a la teoría de conjuntos un carácter fundacional, que de alguna manera

genera problemas en aspectos lógicos como decidir cuando un conjunto "de elección" existe. O cuando un "cardinal" existe[1]. La teoría de tipos de alguna manera evita esto porque habla de cosas que "pueden ser construídas" (sintácticamente). La teoría de tipos define un lenguaje formal, a través del cual construye objectos, que de poder ser representados en dicho lenguaje de forma consistente, esto asegura su existencia. Es decir, a pesar de no tener un carácter fundacional entonces, si nos permite hablar de *fundamentos de las matemáticas*, en términos de que lo que *existe o no* se refiere a si *puede ser construído o no*.

Es de hecho posible intentar construir un tipo que constituya una proposición falsa. Lo que nos lleva a la propiedad mas notable de la teoría de tipos. Si quisiéramos por ejemplo escribir la proposición (i.e definir el tipo): "$x + x = x$", y logramos dar una serie de tipos que "combinados" nos permiten escribir un elemento que sea de ese tipo, entonces tal proposición sería un *teorema*. De esta manera podemos hablar de un teorema como una proposición que fue demostrada. O en términos de la teoría de tipos; un teorema es una proposición que puede ser construida (i.e expresada a través de otros tipos). Por tanto la proposición $x + x = x$ puede no tener sentido su uno no dice "donde" se da esa igualdad, ya que como veremos mas adelante el tipo de igualdades (si, las igualdades son tipos) es distinto para cada tipo donde esta se da. Por ejemplo, dada la sentencia "a,b:C" (i.e a y b son puntos en el tipo C) entonces existe el tipo "$a =_C b$", que en principio podría tener o no algun término, o lo qe es lo mismo, tener o no una prueba y que la igualdad fuera o no cierta. De todas formas es necesario establecer para determinadas ocasiones una igualdad que se comporte de manera similar a la igualdad usual. Es decir, que de manera preestablecida constituya una igualdad, de la misma forma que "$a : A$" es una sentencia "atómica" e innegable en la gramática. Esta igualdad se denota por $\equiv$, y no constituye un tipo en el sentido de que la expresion $a \equiv b : C$ se manifiesta mas como "**a** es igual a **b** por definición" que "**a** es igual a **b**". La segunda es una proposición sujeta a demostración. De manera similar a la expresión ":", el símbolo "$\equiv$" no puede ser usado para negar o probar. Es decir, la sentencia "si **a**$\equiv$ **b** entonces no **c**$\equiv$**d**", no tiene sentido en la teoría.

Los conjuntos además son *extensionales*, es decir, dos conjuntos son iguales si contienen los mismos elementos. Los tipos por otro lado son "intensionales"[2], es decir son iguales si se "escriben" igual. Lo que nos lleva a lo mismo, el conjunto requiere una prueba, pero la prueba del tipo esta dada en su construcción. Decir como

[1] Estan los cardinales inaccesibles: Conjuntos X tales que $\forall Y$ con $|Y| < |X|$, se tiene $|2^Y| < |X|$.
[2] Al menos la teoría que describimos aquí lo es.

en el ejemplo anterior escribiendo $4 + (8 * 3)^2$ estamos dando una prueba de que $4 + (8 * 3)^2 : \mathbb{N}$.

Los conjuntos tienen la característica de que $\in$ es *indecidible*, ya que $x \in X$ (donde x no es variable asumida) requiere una prueba. Mientras $x : X$ es *decidible* ya que requiere un método basado en solamente verificar la sintáxis, que puede ser traducido en verficar las reglas constructivas de x, que se traduce en un "algoritmo de tipado". Esto tiene un inconveniente claro (que es desde el punto de vista de algunos teóricos una limitación) y es el hecho de que por ejemplo

$$\frac{1}{2}\sum_{n:\mathbb{N}} 2^{-n} : \mathbb{N}$$

no esta bien tipado, puesto que uno necesita saber que $\sum_{n:\mathbb{N}} 2^{-n}$ es divisible entre 2, cosa que no está codificada en ninguna de las operaciones de la suma infinita descrita. O regresando al ejemplo del teorema de Fermat, uno podría decir

$$(n, p) : \{n \in \mathbb{N} \mid \exists x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n)\}$$

pudiéndose comprobar que esta bien tipado si tuviéramos una prueba p bien tipada (escrita en una sintáxis correcta)

$$p : \exists x, y, z \in \mathbb{N}^+ (x^n + y^n \neq z^n).$$

Por todo lo anterior es claro que no podemos hablar de un término "$a : A$" (i.e un punto en un tipo) sin enunciar su tipo correspondiente. De igual forma no podemos plantear expresiones o crear tipos de la forma: "$a : A$ prueba que no ocurre $b : B$". Las variables son capaces de existir solo después de que enunciamos la existencia (i.e escribimos) de su tipo.

Para resumir un poco todo lo anterior veamos un ejemplo algo informal. Supongamos que tenemos una función $f : \mathbb{N} \longrightarrow \mathbb{N}$, donde aquí los dos puntos significan que la función f es un término del tipo $\mathbb{N} \longrightarrow \mathbb{N}$ (i.e las funciones de los naturales en los naturales). Entonces si pudieramos decir que f esta definida por la equación $f(x) = x^2$ (y esta es la parte informal del ejemplo), entonces $f(2) \equiv 2^2$ por definición. De esta forma si tenemos una prueba $p : 2^2 =_{\mathbb{N}} 4$, entonces es válido escribir $p : f(2) =_{\mathbb{N}} 4$, ya que por definición $f(2)$ **es** 2^2. Todo lo anterior es equivalente a decir que las sentencias $p : 2^2 =_{\mathbb{N}} 4$ y $f(2) \equiv 2^2$ *derivan* la sentencia $p : f(2) =_{\mathbb{N}} 4$.

En el ejemplo anterior usamos la expresion $f(x) = x^2$ para "definir" la función f. Sin embargo en teoría de tipos con frecuencia es necesario dar la *definición* de un término en un tipo mediante una expresión. Así se introduce el símbolo ":≡". Que si es usado en el párrafo anterior nos da una definición totalmente válida de f y al escribir $f(x) :\equiv x^2$, el ejemplo queda formalizado.

El paradigma de la teoría tipada queda claro (como casi todo en la matemática) cuando uno se vuelve familiar con el. Así que empezar a construir la teoría de manera formal es la mejor forma de introducir cada uno de estos conceptos.

2.1.2. Tipos de funciones

Los primeros tipos que presentaremos son los tipos de funciones. No por ser los mas básicos si no por ser los mas ilustrativos en la abstracción de la teoría. De manera natural entonces si tenemos dos tipos A y B definimos el **tipo** de las funciones (o mapeos) con dominio A y codominio B, denotado $A \longrightarrow B$. A diferencia de lo que ocurre en la teoría de conjuntos esto es un concepto primitivo. Es decir el tipo de funciones se diferencia del conjunto de funciones en que dados dos conjuntos es necesario declarar la funcion para poder asegurar que en efecto su codominio es B. Pero en teoría de tipos el tipo de las funciones existe (tiene al menos una prueba que es una funcion constante cualquiera para cualquier $b : B$), por tanto uno puede hablar de elementos $f : A \longrightarrow B$ sin siquiera declarar explicitamente la forma de la función. O lo que es lo mismo, lo único que debemos saber es que para todo punto $f : A \longrightarrow B$, se cumple que si $a : A$ entonces $f(a) : B$. En cualquier caso es muy comun en la teoría de tipos usar una **lambda abstracción** para definir de manera explicita los mapeos entre tipos. Esto es, dada una función $f : A \longrightarrow B$, podemos definir: $f(x) :\equiv \Lambda$, donde el termino x se comporta como una variable y Λ se comporta como un objeto abstracto que toma valores (o se **evalúa**) dependiendo del que asuma x pero que en cualquier caso $\Lambda(x) : B$. Si volvemos a nuestro ejemplo anterior, tomamos

$$f : \mathbb{N} \longrightarrow \mathbb{N}$$

y podemos hacer $f(x) :\equiv x^2$ entonces es $f(2) \equiv 2^2$ como ya se vió, pero tal vez no querríamos introducir la definicion explícita de dicha función, así que simplemente hacemos $f(x) :\equiv \Lambda$ y tenemos que $\Lambda : B$, por tanto

$$(\lambda(x : A).\Lambda) : A \longrightarrow B$$

que en nuestro ejemplo se traduce en

$$(\lambda(x : A).x^2) : A \longrightarrow B$$

lo que permite trabajar con abstracciones de una función sin tener una definición explícita. Por ultimo vale la pena hacer notar que el **contexto** de la variable no siempre se declara. Es decir $(\lambda(x : A).x^2)$ en general se ecribe como $\lambda x.x^2$, lo que significa que lo que esta escrito despues del punto es la declaracion de la lambda abstracción. Otra cosa que vale la pena notar es que usando nuestras reglas hasta el momento podemos escribir

$$(\lambda x.x^2)(x) \equiv 2^2$$

lo que completa el ejemplo.

Hay otra cosa que es importante sobre los tipos de funciones y es el hecho de porque el parámetro que está "abstraído" lo llamamos variable. Para esto pongamos un ejemplo. Supongamos que tenemos la siguiente función

$$f : \mathbb{N} \longrightarrow (\mathbb{N} \longrightarrow \mathbb{N})$$

esta es una función que para elemento de los naturales asigna una función en los naturales. Es decir esta podría ser declarada por ejemplo como la suma por un número: $f(x) :\equiv \lambda y.x + y$. Así para cada valor de x tenemos una función $\lambda y.x + y$ que a cada valor de $y : \mathbb{N}$ le suma x (que ya esta fijo). A estas alturas debe ser claro que si x no se encuentra en la definición entonces no tiene ningun efecto en la lambda abstracción. Es decir si $f(x) :\equiv \lambda y.y^2$, entonces da igual valor de x, siempre tendremos $(\lambda y.y^2)(x) \equiv \lambda y.y^2$. Aún así es importante notar que en la definición de la lamda abstracción la variable en uso es indistinta, lo que importa es cual es la letra usada en la abstracción. Así, podemos escribir las siguientes afirmaciones

$$\lambda y.y^2 \equiv \lambda z.z^2,$$

$$(\lambda y.y + x)(2) \equiv (\lambda z.z + x)(2), \textit{ y que}$$

$$\lambda y.\lambda x.y^2 + x \equiv \lambda z.\lambda x.z^2 + x$$

En la última de las afirmaciones tenemos una doble abstracción (o una funcion de dos variables). Lo que de igual forma nos dice que en el caso de la segunda hay que tener cuidado al hacer $(\lambda y.y + x)(x)$.

Para fijar ideas en la notación, en el último de los ejemplos podemos notar que si definimos $f :\equiv y^2 + x \equiv \lambda y.\lambda x.y^2 + x$, entonces f también puede escribirse como $f(x)(y)$ ó $f(x, y)$, y esta dada por la expresion $f(x, y) :\equiv y^2 + x$. También se escribe en ocasiones $\lambda x.\lambda y.f(x, y)$

2.1.3. Universos y Familias de tipos

Los universos son simplemente tipos tales que sus elementos son a la vez tipos. La diferencia en este punto entre la teoria de conjuntos y la de tipos es que para evitar las paradojas que resultan de las autoreferencias (conjuntos que se contienen a si mismos), se introduce una gerarquía entre los universos. Es decir ya que los universos son tipos que contienen tipos, entonces hay un orden jerárquico tal que

$$\mathcal{U}_0 : \mathcal{U}_2 : \mathcal{U}_2 : \dots$$

El orden es parcial, porque solo podemos decir para cada tipo en que universo esta contenido. Además, tenemos que es un orden acumulativo pues si el tipo $A : \mathcal{U}_k$ entonces $A : \mathcal{U}_{k+1}$. Dicha gerarquía es algo implícita, así que en general no usaremos subíndices en nuestra notación, y no nos preocuparemos por la gerarquía de los universos mencionados en este texto. Así que en general denotaremos al tipo $A : \mathcal{U}$, donde $\mathcal{U}$ es algún universo que contenga a A (se podría pensar en el mas chico, pero esto implicaría buscar su existencia y ni siquiera es necesario). Una teoría mas formal con sus argumentos correspondientes sobre los universos de tipos puede ser encontrada en [4].

Introducidos los universos podemos entonces hablar de funciones entre tipos y universos, es decir $f : A \longrightarrow \mathcal{U}$. Esto es, una función que asigna cada elemento en A un tipo en $\mathcal{U}$. A estos tipos se les llama **familias** o **tipos dependientes**. El ejemplo clásico de un objeto de este tipo es la familia $Fin : \mathbb{N} \longrightarrow \mathcal{U}$. Donde a cada elemento $n : \mathbb{N}$ (obviamos aqui por un momento la definición de $\mathbb{N}$ que aún no presentamos formalmente) se le asigna un conjunto finito de "orden" n, $Fin(n) : \mathcal{U}$. Es decir si denotamos a los elementos de $Fin(n)$ por $0_n, 1_n, ..., (n-1)_n$, entonces para dos índices distintos j, k, tenemos que 0_j es distinto de 0_k, ya que tienen distintos tipos ($Fin(j)$ es un tipo distinto a $Fin(k)$).

Un ejemplo de algo que no funciona como una familia sería por ejemplo tratar de escribir $\lambda(i : \mathbb{N}).\mathcal{U}_i$, pero esto sería algo de tipo $\mathbb{N} \longrightarrow \mathcal{U}$, donde $\mathcal{U}$ tendría que ser lo suficientemente grande para contener a todos los universos. Tal universo no existe.

2.1.4. Funciones dependientes ($\prod$-tipos)

Las funciones dependientes son similares a las familias solo que su codominio varía sobre una familia de tipos. Es decir, dado un tipo $A : \mathcal{U}$ y una familia $B : A \longrightarrow \mathcal{U}$, podemos construir

$$f : \prod_{(x:A)} (B(x) : U)$$

o si usamos la λ- abstracción:

$$\lambda(x : A).\Lambda : \prod_{x:A} B(x)$$

Es decir, $f(a) : B(a)$, o mejor dicho, f es una función tal que para cada término $x : A$ tenemos un elemento en el tipo $B(x)$, es decir el codominio de la función toma valores en un tipo específico de $\mathcal{U}$ para cada x. Como ejemplo veamos lo siguiente. Siempre podemos introducir la familia constante $C : A \longrightarrow \mathcal{U}$ donde $B : \mathcal{U}$ y $C(x) \equiv B$ para cualquier $x : A$. Entonces esta familia consta de un solo elemento que es B. Usando la notación lambda sería $\lambda(x : A).B : A \longrightarrow \mathcal{U}$ por eso en estos casos la familia se denota por $B(x) \equiv B$. Usando el ejemplo vemos que pasa con la funcion dependiente

$$\prod_{x:A} C(x) : \mathcal{U} \equiv \prod_{x:A} B : \mathcal{U}$$

por tanto el codominio siempre es constante y tenemos que

$$\prod_{x:A} C(x) : \mathcal{U} \equiv A \longrightarrow B$$

Hay algunas otras funciones que son definidas usando la noción de familias de tipos y son las llamadas polimorfas. El ejemplo mas trivial es el de la identidad

$$id :\equiv \prod_{A:\mathcal{U}} A \longrightarrow A$$

el codominio de la funcion son las funciones $A \longrightarrow A$ para cada $A : \mathcal{U}$. Es decir su codominio esta formado por todas las funciones identidades de elementos del

universo. Es de esta forma una identidad que esta en un "nivel superior" de abstracción en el sentido de que su codominio contiene elementos $id_A(x) :\equiv x$ donde cada $A : \mathcal{U}$ nos da una función identidad distinta. Esta función también se puede escribir como $id :\equiv \lambda(A : U).\lambda(x : A).x$.

En un ejemplo de función polimorfa tenemos la funcion **swap** que es

$$\mathbf{swap} : \prod_{(A:\mathcal{U})} \prod_{(B:\mathcal{U})} \prod_{(C:\mathcal{U})} (A \to B \to C) \to (B \to A \to C)$$

o que se escribe tambien

$$\mathbf{swap} : \prod_{A,B,C:\mathcal{U}} (A \to B \to C) \to (B \to A \to C)$$

que usando notación lambda se define como

$$\mathbf{swap}(A, B, C, h) :\equiv \lambda b.\lambda a.h(a, b)$$

o

$$\mathbf{swap}_{A,B,C}(h)(b, a) :\equiv \lambda b.\lambda a.h(a, b)$$

donde los subíndices representan los tipos que son argumentos de dependencia. Las cosas podrían complicar un poco mas la notación ya que en este caso los tipos no dependen uno del otro. Pero si por ejemplo $A : \mathcal{U}$, $B : A \longrightarrow \mathcal{U}$ y $C : \prod_{x:A} B(x) \longrightarrow \mathcal{U}$, entonces tendríamos el tipo

$$\prod_{x:A} \prod_{y:B(x)} C(x, y)$$

al cual no podemos aplicarle la función **swap**. En caso de tener $f : \prod_{(x:A)} \prod_{(y:B(x))} C(x, y)$, podríamos escribir como anteriormente hicimos informalmente $f(x, y) :\equiv C(x, y)$.

2.2. Mecanismos de generación de tipos

Cuando queremos introducir un tipo nuevo, en general hay un procedimiento estándar para establecer las características del tipo que estamos definiendo. Es decir, debemos enunciar sus características y patrones de interacción con otros tipos, de forma que esté bien definido.

- **Reglas de formación.** Con las reglas de formación nos referimos al contexto en el cual es válido generar este tipo. Por ejemplo; dado tipos A y B, podemos definir el tipo $A \longrightarrow B$.

- **Reglas de introducción.** Una regla de introduccion, o **constructor**, es aquella que permite definir un tipo a través de una sintaxis establecida. Por ejemplo las λ-abstracciones permiten introducir funciones. Es decir, si definimos: $f(x) :\equiv x + 2$, entonces $f(x) :\equiv \lambda x.x + 2$.

- **Reglas de eliminación.** Las reglas de eliminación son aquellas que definen como el tipo actúa de forma abstracta. Un ejemplo de esto es la capacidad de una función de "ser aplicada".

- **Reglas de computación.** Las reglas de computación (también llamadas β-reducción) definen como el tipo interactúa con la sintáxis. O dicho de otra forma, como las reglas de eliminación actúan sobre el constructor. Es decir, como se puede reemplazar la sintaxis actual por algun símbolo distinto dado que el tipo esta actuando de determinada forma. Un ejemplo de esto es la evaluación en las lambda abstracciones, es decir, dado el símbolo $f(x) :\equiv \lambda x.\Phi$, declaramos que $f(z)$ es $(\lambda x.\Phi)(z)$, que significa reemplazar z en cada lugar de Φ, donde exista una x.

- **Principio de unicidad.** El principio de unicidad (conocido como η-expansión) establece que cada elemento en el tipo puede ser determinado de manera unívoca, a través de la aplicación sucesivas de pasos de eliminación sobre su representación actual. De manera dual a las reglas de computación, la η-exapansión define como los constructores actúan en las reglas de eliminación. Es decir, si la regla de eliminación de una función es la capacidad de "ser aplicada". Entonces el principio de unicidad establece que dados los términos de introducción de un elemento en el tipo, entonces la función esta determinada por los valores de computación sobre cada uno de ellos.

2.3. Tipos de productos

Para definir el tipo de productos supongamos que queremos aplicar una funcion f a algun tipo de estos, es decir, queremos saber como se computa $f : A \times B \longrightarrow C$. Denotemos por (a, b) los elementos del tipo $A \times B$. En otras palabras, supongamos que

queremos definir como computar $f(a, b) : C$. Para esto, nos auxiliamos de una funcion $g : A \longrightarrow B \longrightarrow C$. Digamos entonces que es posible definir $f(a, b) :\equiv g(a)(b)$. Esta es una regla de eliminación, pues define como se aplica una función que toma valores en un tipo producto, ya que g está siendo evaluada en tipos ordinarios (no quiere decir que los productos sean extraordinarios :)).

Vale la pena notar la diferencia sustancial en la manera de defnir un conjunto y un tipo. La teoría de conjuntos define el conjunto $A \times B$ como la colección de pares ordenados (a, b) donde $a \in A$ y $b \in B$. La teoría de tipos en cambio, establece que una función esta bien definida si podemos computarla en pares ordenados (a, b) donde $a : A$ y $b : B$. Dicho esto, si un elemento x puede ser evaluado en f, entonces eso es una *prueba* de que $x : A \times B$.

La construcción anterior debería ser realizada cada vez que quisieramos definir un producto $A \times B$. Para evitar esto se define una estructura (un tipo) que permita "automatiza" dicha construcción. Así tenemos

$$rec_{A\times B} : \prod_{C:\mathcal{U}} (A \longrightarrow B \longrightarrow C) \longrightarrow A \times B \longrightarrow C$$

con la siguiente ecuación como definición

$$rec_{A\times B}(C, g, (a, b)) :\equiv g(a)(b)$$

La función $rec_{A\times B}$ es conocida como **recursor**. El sentido que tiene es simplemente establecer el hecho de que es posible definir una función $f : A \times B \longrightarrow C$ dando sus valores en los pares ordenados (a, b).

La definición que hemos dado para la función f no permite definir funciones dependientes sobre el tipo producto. Para esto necesitamos extender el concepto de recursor. Dado $C : A \times B \longrightarrow \mathcal{U}$, podemos definir el tipo $f : \prod_{x:A\times B} C(x)$, a través de una función $g : \prod_{a:A} \prod_{b:B} C((a, b))$, usando la siguiente ecuación como definición

$$f(x) :\equiv g(a)(b)$$

Por ejemplo, usando esto podemos probar el principio de unicidad, que en este caso establece que cada elemento $x : A \times B$ es un par ordenado. Primero notamos que (sin dar una definición formal), para cada tipo A, tenemos el tipo $(=_A)$. Y tenemos además

un elemento $refl_x : x =_A x$. Entonces dadas

$$pr_1((a,b)) :\equiv a$$

$$pr_2((a,b)) :\equiv b$$

podemos construir la función

$$uppt : \prod_{x:A\times B} ((pr_1(x), pr_2(x)) =_{A\times B} x)$$

como $(pr_1(a,b), pr_2(a,b)) \equiv (a,b)$ por definición. Entonces sabemos, usando el juicio anterior, que

$$refl_{(a,b)} : (pr_1(a,b), pr_2(a,b)) =_{A\times B} (a,b)$$

está bien tipado.

De manera un poco mas general, la construcción anterior implica que para definir una función dependiente en un producto, solo debemos definir sus valores en cada uno de los vectores canónicos. De esta manera definimos el **inductor** de tipos producto, dados $A, B : \mathcal{U}$

$$ind_{A\times B} : \prod_{C:A\times B\to\mathcal{U}} (\prod_{a:A}\prod_{b:B} C((a,b))) \longrightarrow \prod_{x:A\times B} C(x)$$

con la siguiente ecuación como definición

$$ind_{A\times B}(C, g, (a,b)) :\equiv g(a)(b)$$

Un ejemplo de inducción, es la del tipo unitario $(* : \mathbf{1})$, que permite demostrar que el tipo unitario $\mathbf{1}$, tiene como único elemento $*$. Para esto vemos que

$$ind_{\mathbf{1}} : \prod_{C:\mathbf{1}\to\mathcal{U}} C(*) \longrightarrow \prod_{x:\mathbf{1}} C(x)$$

con la siguiente ecuación como definición

$$ind_{\mathbf{1}}(C, c, *) :\equiv c$$

Y entonces, tomando

$$upun : \prod_{x:\mathbf{1}} x = *$$

$$C :\equiv \lambda x.x$$

por la definición tenemos que si $* : \mathbf{1}$

$$upun(*) :\equiv refl_*$$

es decir $* =_{\mathbf{1}} *$.

2.4. Productos dependientes

Es posible hacer una construcción similar a la que se hizo con las funciones dependientes, estos son los llamados tipos de **productos dependientes** (tambien llamados pares dependientes). Es decir, productos donde la segunda coordenada varía dependiendo del valor de la primera. Estos productos se denotan como $\sum_{x:A} B(x)$. Esta notación se refiere al hecho de que si $(a, b) : \sum_{x:A} B(x)$, esto quiere decir que $a : A$ y $b : B(a)$. Por tanto si el tipo B fuera constante, tendríamos el producto definido anteriormente, esto es $\sum_{x:A} B \equiv A \times B$. Por lo anterior es claro que podemos definir las proyecciones del tipo $\sum_{x:A} B(x)$ de la siguiente forma

$$p_1 : \sum_{x:A} B(x) \longrightarrow A$$

se define por

$$p_1(a, b) :\equiv a$$

Sin embargo la definición de p_2 es algo mas intrincada, ya que como en el segundo tipo depende del primero, entonces p_2 deberá ser una función dependiente

$$p_2 : \prod_{p:\sum_{x:A} B(x)} B(p_1(p))$$

Para ver que esto esta bien tipado, veamos como podemos definir una función del tipo $F : \sum_{x:A} B(x) \longrightarrow \mathcal{U}$. Para esto tendríamos que tener una definición

$$g : \prod_{(a:A)} \prod_{(b:B(a))} F((a, b))$$

de la cual definiríamos

$$f : \prod_{p:\sum_{x:A} B(x)} F(p)$$

de la manera obvia, como $f((a,b)) :\equiv g(a)(b)$. Asi tomando $F(p) :\equiv B(p_1(p))$, por lo que definimos anteriormente, tenemos

$$p_2 : \prod_{p:\sum_{x:A} B(x)} B(p_1(p))$$

lo que hace ver de manera obvia

$$p_2((a,b)) :\equiv g(a)(b) :\equiv b$$

pues $g : \prod_{(a:A)} \prod_{(b:B(a))} B(p_1(p))$. De manera muy similar a como hicimos anteriormente podermos definir el recursor de la siguiente forma

$$rec_{\sum_{x:A} B(x)} : \prod_{C:\mathcal{U}} \Big(\prod_{x:A} B(x) \Big) \longrightarrow \big(\sum_{x:A} B(x) \big) \longrightarrow C$$

definido por la ecuacion

$$rec_{\sum_{x:A} B(x)}(C, g, (a,b)) :\equiv g(a)(b)$$

De manera similar, tenemos el inductor del tipo $\sum_{x:A} B(x)$ como:

$$ind_{\sum_{x:A} B(x)} : \prod_{(F:\sum_{x:A} B(x) \longrightarrow \mathcal{U})} \Big(\prod_{(a:A)} \prod_{(b:B(a))} F((a,b)) \Big) \longrightarrow \prod_{(p:\sum_{x:A} B(x))} F(p)$$

definido por la ecuación

$$ind_{\sum_{x:A} B(x)}(F, g, (a,b)) :\equiv g(a)(b)$$

Como un ejemplo de lo anterior consideremos el caso donde A y B son tipos, y $R : A \longrightarrow B \longrightarrow \mathcal{U}$. Definimos

$$ac : \Big(\prod_{x:A} \sum_{y:B} R(x,y) \Big) \longrightarrow \Big(\sum_{(f:A \longrightarrow B)} \prod_{a:A} R(x, f(x)) \Big)$$

En estos casos R es llamada una **relación demostrativa** (en inglés: proof-relevant relation), en el sentido de que la existencia de un elemento en este tipo demuestra alguna proposición. Notemos que ac dice de manera intuitiva que si tenemos una función dependiente que asigna a cada elemento $a : A$ un par (b, r), donde $b : B(a)$ y $r : R(a, b)$, entonces esto nos induce una función $f : A \longrightarrow B$ de manera que a cada $a : A$, le asignamos $(f(a), r)$, donde $r : R(a, f(a))$, que verifica la relación. En otras palabras, si sustituimos el símbolo $\prod$ por "para todo" y el símbolo $\sum$ por "existe", tenemos la sentencia: Si para todo $a : A$ existe un $b : B$ que verifica $R(a, b)$, entonces existe una función $f : A \longrightarrow B$ tal que para todo $a : A$ se cumple que $R(a, f(a))$. La sentencia anterior es algo equivalente a una de las representaciones del teorema de elección, por ello, es llamado **teorema tipado de elección**. Sucede que en este caso la prueba surge de manera natural por las propiedades constructivas de la teoría tipada, mas que un axioma asumido para evitar alguna paradoja lógica. Realmente no hay elección alguna sobre elementos de un tipo (menos aún de un conjunto, colección, etc), solo que la parte del tipo representada por el origen de la flecha representa la elección del elemento adecuado $b : B$, y el destino de la flecha verifica la consistencia (que sea bien tipado) de la conclusión de la proposición que esta codificada en el tipo.

2.5. Magmas

Los magmas son tipos definidos como (A, m) donde $A : \mathcal{U}$ es un tipo y

$$m : A \longrightarrow A \longrightarrow A$$

es una operación binaria en A. Es obvio que son un par ordenado dependiente, es decir pueden escribirse como

$$magma :\equiv \sum_{A:\mathcal{U}} \big(A \longrightarrow A \longrightarrow A\big)$$

dado un magma podemos usar las proyecciones p_1 y p_2 para extraer su tipo "base" y su operación respectivamente. Existen además los magmas punteados

$$pmagma :\equiv \sum_{A:\mathcal{U}} \big(A \longrightarrow A \longrightarrow A\big) \times A$$

es decir son tipos $(A, (m, e))$ donde $e : A$ es el/un punto básico de A. La disposición de los parentésis de alguna manera indica que el punto $e : A$ puede depender de m.

Aunque en general sean dependientes o no la notación de n tuplas se escribira de manera anidada, esto es $(x, y, z, w, \dots) :\equiv (x, (y, (z, (w, \dots))))$.

2.6. Tipos de coproductos

Dados dos tipos A y B denotamos su **coproducto** como $A + B : \mathcal{U}$. A pesar de que no existe una definición como "unión ajena" de tipos, esta está representada por el coproducto. De esta forma para construir elementos en $A + B$, estos deben estar en A o estar en B. Es decir, tenemos "inclusiones"

$$inl : A \longrightarrow A + B$$

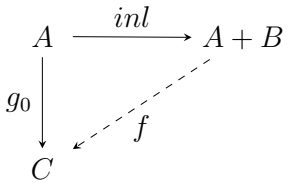
$$inr : B \longrightarrow A + B$$

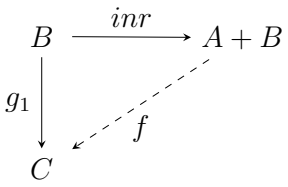
Para construir funciones $f : A + B \longrightarrow C$ necesitamos operadores

$$g_0 : A \longrightarrow C$$

$$g_1 : B \longrightarrow C$$

entonces podemos definir f a través de los diagramas





Esto es, la fiunción f esta definida por casos

$$f(inl(a)) :\equiv g_0(a)$$

$$f(inr(b)) :\equiv g_1(b)$$

En una construcción similar a las anteriores podemos definir el recursor

$$rec_{A+B} : \prod_{C:\mathcal{U}} (A \longrightarrow C) \longrightarrow (B \longrightarrow C) \longrightarrow A + B \longrightarrow C$$

Definido por las ecuaciones

$$rec_{A+B}(C, g_0, g_1, inl) :\equiv g_0(a)$$

$$rec_{A+B}(C, g_0, g_1, inr) :\equiv g_1(b)$$

Es conveniente introducir ahora el tipo nulo o vacío, es decir $\mathbf{0} : \mathcal{U}$. Podemos decir entonces que como $\mathbf{0}$ no tiene algun elemento, cualquier función $f : \mathbf{0} \longrightarrow C$ no necesita ecuación definitoria alguna. El recursor entonces es

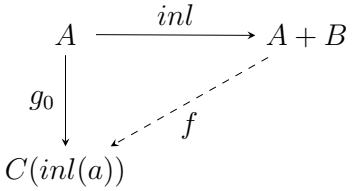
$$rec_{\mathbf{0}} : \prod_{\mathbf{0}:\mathcal{U}} (\mathbf{0} \longrightarrow C)$$

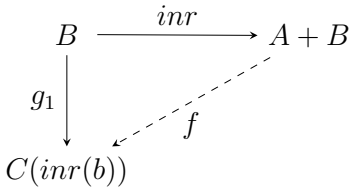
esto es una función trivial del $\mathbf{0}$ en cualquier otro tipo. Lógicamente representa el hecho de que todo lo que exista en $rec_{\mathbf{0}}$ para un tipo $C : \mathcal{U}$, es verdadero. Para construir funciones dependientes del coproducto, es decir el tipo $\prod_{q:A+B} C(q)$ necesitamos definir g_0 y g_1 adecuadas, es decir dependientes, hacemos

$$g_0 : \prod_{x:A} C(inl(x))$$

$$g_1 : \prod_{y:B} C(inr(y))$$

de esta forma tomando inl, inr como antes, tenemos los diagramas correspondientes





que se definen mediante las ecuaciones

$$f(inl(a)) :\equiv g_0(a)$$

$$f(inr(b)) :\equiv g_1(b)$$

De la misma forma podemos definir el inductor del tipo de coproductos

$$ind_{A+B} : \prod_{(F:A+B \longrightarrow \mathcal{U})} \Big(\prod_{a:A} F(inl(a))\Big) \longrightarrow \Big(\prod_{b:B} F(inr(b))\Big) \longrightarrow \Big(\prod_{q:A+B} F(q)\Big)$$

Igual que anteriormente si F es constante coincide con la definición del recursor. El inductor del elemento nulo $\mathbf{0} : \mathcal{U}$ es

$$ind_{\mathbf{0}} : \prod_{F:\mathbf{0}\to\mathcal{U}} \left(\prod_{z:\mathbf{0}} F(z)\right)$$

2.7. El tipo booleano

El tipo booleano, denotado $\mathbf{2} : \mathcal{U}$, es el tipo que contiene exctamentedos elementos, digánse $0_{\mathbf{2}}, 0_{\mathbf{1}} : \mathcal{U}$. Para definir una función $f : \mathbf{2} \longrightarrow C$, necesitamos dos valores $c_0, c_1 : C$, así definimos f mediante las ecuaciones

$$f(0_{\mathbf{2}}) :\equiv c_0$$

$$f(1_{\mathbf{2}}) :\equiv c_1$$

El recursor se define de manera obvia usando el hecho anterior, necesitamos dos valores para definir la función

$$rec_{\mathbf{2}} :\equiv \prod_{C:\mathcal{U}} C \longrightarrow C \longrightarrow \mathbf{2} \longrightarrow C$$

definido por las ecuaciones

$$rec_{\mathbf{2}}(C, c_0, c_1, 0_{\mathbf{2}}) :\equiv c_0$$

$$rec_{\mathbf{2}}(C, c_0, c_1, 1_{\mathbf{2}}) :\equiv c_1$$

El principio anterior representa un control de flujo algorítmico, es decir, puede verse como una sentencia algorítmica de tipo **if-then-else**. Esto se puede parafrasear de la siguiente forma: dado $0_{\mathbf{2}}$ entonces c_0, en caso contrario (se cumple $1_{\mathbf{2}}$) entonces c_1. Este hecho se ve claramente representado en la sintáxis del recursor. Es por ello que el tipo $\mathbf{2}$ obtiene su nombre de "tipo booleano". Para construir funciones dependientes solo debemos notar que en tal caso c_0 y c_1 han de estar en un tipo dependiente, esto es $f : \prod_{x:\mathbf{2}} C(x)$ y

$$c_0 : C(0_{\mathbf{2}}), f(0_{\mathbf{2}}) :\equiv c_0$$

$$c_1 : C(1_{\mathbf{2}}), f(1_{\mathbf{2}}) :\equiv c_1$$

De esta forma tenemos el principio de inducción

$$ind_{\mathbf{2}} :\equiv \prod_{(F:\mathbf{2}\to\mathcal{U})} F(0_{\mathbf{2}}) \longrightarrow F(1_{\mathbf{2}}) \longrightarrow \prod_{x:\mathbf{2}} C(x)$$

definido por las ecuaciones

$$ind_{\mathbf{2}}(F, 0_{\mathbf{2}}, 1_{\mathbf{2}}, c_0) :\equiv c_0$$

$$ind_{\mathbf{2}}(F, 0_{\mathbf{2}}, 1_{\mathbf{2}}, c_1) :\equiv c_1$$

Cuando hablamos de los $\sum$-tipos mas arriba dijimos que de alguna manera eran uniones disjuntas indexadas por los elementos de un tipo, y cuando mencionamos los coproductos los definimos como uniones disjuntas binarias. Es decir, que el coproducto $A+B$ puede verse como $\sum$-tipo indexado sobre un tipo binario (con dos elementos), es decir $\mathbf{2}$. De manera un poco mas clara, podemos definir una familia $F : \mathbf{2} \to \mathcal{U}$, donde $F(0_{\mathbf{2}}) \equiv A$ y $F(1_{\mathbf{2}}) \equiv B$. Para esto podemos usar el hecho de que $\mathcal{U}$ es un tipo, y $A, B : \mathcal{U}$, entonces a través de $rec_{\mathbf{2}}$ tenemos

$$A + B :\equiv \sum_{x:\mathbf{2}} rec_{\mathbf{2}}(\mathcal{U}, A, B, x)$$

ya que $rec_{\mathbf{2}}(\mathcal{U}, A, B, 0_{\mathbf{2}}) \equiv A$ y $rec_{\mathbf{2}}(\mathcal{U}, A, B, 1_{\mathbf{2}}) \equiv B$. La definición anterior tiene como ecuaciones definitorias

$$inl(a) :\equiv (0_{\mathbf{2}}, a)$$

$$inr(b) :\equiv (1_{\mathbf{2}}, b)$$

Una idea similar puede ser aplicada a los productos y los $\prod$-tipos. Es decir, podemos definir un producto como

$$A \times B :\equiv \prod_{x:\mathbf{2}} rec_{\mathbf{2}}(\mathcal{U}, A, B, x)$$

definido por ecuaciones

$$pr_1(p) :\equiv p(0_{\mathbf{2}})$$

$$pr_2(p); \equiv p(1_{\mathbf{2}})$$

Vale la pena notar de que esta construcción de familias de tipos a través de recursores o inductores y haciendo uso del tipo $\mathcal{U}$ es una técnica muy útil para la teoría ya que esta es netamente constructiva.

2.8. El tipo de los números naturales

El tipo de los números naturales es infinito, donde con infinito nos referimos a que es posible construir una cantidad ilimitada de elementos en él. Los elementos de $\mathbb{N} : \mathcal{U}$ son construídos a partir de el elemento $0 : \mathcal{U}$ y una función $succ : \mathbb{N} \longrightarrow \mathbb{N}$. Los números naturales los denotaremos con la notación decimal usual

$$1 :\equiv succ(0), 2 :\equiv succ(1), 3 :\equiv succ(2), \ldots$$

En los números naturales los conceptos de recursión e induccón tienen un significado mas tradicional. Por ejemplo, para definir una función $f : \mathbb{N} \longrightarrow C$, necesitamos un elemento inicial $c_0 : C$ y una función de recursión $c_s : C \longrightarrow C$. Así se define f como

$$f(0) :\equiv c_0$$

$$f(succ(n)) :\equiv c_s(n, f(n))$$

Como un ejemplo veamos la funcion $double : \mathbb{N} \longrightarrow \mathbb{N}$, es decir la función que le asigna a cada natural su doble. Primero tomamos por razones obvias $double(0) :\equiv 0$. Luego necesitamos definir

$$c_s(succ(n), f) :\equiv succ(succ(f(n)))$$

de esta manera tenemos

$$double(succ(n)) \equiv succ(succ(double(n)))$$

Por ejemplo, si quisiéramos encontrar $double(1)$

$$double(1) \equiv double(succ(0)) \equiv succ(succ(double(0))) \equiv succ(succ(0)) \equiv 2$$

Usando el esquema anterior podemos definir funciones multivariadas tomando C como una función. Por ejemplo podemos definir la función suma $add : \mathbb{N} \longrightarrow \mathbb{N} \longrightarrow \mathbb{N}$, donde estamos tomando $C : \mathbb{N} \longrightarrow \mathbb{N}$. Entonces tenemos las siguientes reglas de definición

$$c_0 : \mathbb{N} \longrightarrow \mathbb{N},$$

$$c_0(0, g)(n) :\equiv g(n)(0) :\equiv n$$

y

$$c_s : \mathbb{N} \longrightarrow (\mathbb{N} \longrightarrow \mathbb{N}) \longrightarrow (\mathbb{N} \longrightarrow \mathbb{N}),$$

$$c_s(succ(m), g)(n) :\equiv succ(g(n))$$

De donde obtenemos $add : \mathbb{N} \longrightarrow \mathbb{N} \longrightarrow \mathbb{N}$ que satisface

$$add(0, n) \equiv n,$$

$$add(succ(m), n) \equiv succ(add(m, n))$$

En este caso hacemos la suma

$$add(2, 2) \equiv succ(add(1, 2)) \equiv succ(succ(add(0, 2))) \equiv succ(succ(2)) \equiv 4.$$

El principio de recursión es ahora mucho mas intuitivo para los naturales

$$rec_{\mathbb{N}} : \prod_{C:\mathcal{U}} C\Big(\mathbb{N} \longrightarrow C \longrightarrow C\Big) \longrightarrow \mathbb{N} \longrightarrow C$$

definido por las ecuaciones

$$rec_{\mathbb{N}}(C, c_0, c_s, 0) :\equiv c_0$$

$$rec_{\mathbb{N}}(C, c_0, c_s, succ(n)) :\equiv c_s(n, rec_{\mathbb{N}}(C, c_0, c_s, n))$$

usando el recursor de los naturales podemos excribir las funciones $double$ y add como

$$double : rec_{\mathbb{N}}(\mathbb{N}, 0, \lambda n.\lambda f.succ(succ(f(n))))$$

$$add : rec_{\mathbb{N}}(\mathbb{N} \longrightarrow \mathbb{N}, \lambda n.n, \lambda n.\lambda m.\lambda g.succ(g(n)(m)))$$

Siguiendo el mismo procedimiento podemos pensar en funciones dependientes como argumentos para extender el principio de recursión al de inducción. Es decir, asumimos $F : \mathbb{N} \longrightarrow \mathcal{U}$, $c_0 : F(0)$ y una función

$$c_s : \prod_{n:\mathbb{N}} F(n) \longrightarrow F(succ(n))$$

, de manera que $f : \prod_{n:\mathbb{N}} F(n)$ se defina por las ecuaciones

$$f(0) :\equiv c_0$$

$$f(succ(n)) :\equiv c_s(n, f(n)).$$

Por todo lo anterior

$$ind_{\mathbb{N}} : \prod_{F:\mathbb{N}\longrightarrow\mathcal{U}} F(0) \longrightarrow \Big(\prod_{n:\mathbb{N}} F(n) \longrightarrow F(succ(n))\Big) \longrightarrow \prod_{n:\mathbb{N}} F(n)$$

definido por las ecuaciones

$$ind_{\mathbb{N}}(C, c_0, c_s, 0) :\equiv c_0$$

$$ind_{\mathbb{N}}(C, c_0, c_s, succ(n)) :\equiv c_s(n, ind_{\mathbb{N}}(C, c_0, c_s, n))$$

La definición anterior muestra que el principio de inducción de la teoría de tipos es exactamente igual. En la teoría de tipos para probar una proposición simplemente "construímos" la proposición y si dicha proposición esta bien tipada significa que esta es la prueba. En otras palabras encontrar un elemento en el tipo es la prueba. En el caso de los naturales si se require probar algo para todo número natural es necesario encontrar pruebas en toda la familia $P : \mathbb{N} \longrightarrow \mathcal{U}$, donde P es una proposición que debe cumplirse para todo $n : \mathbb{N}$. Sin embargo el principio de inducción aterior deja claro por su misma definiciíon que si tenemos $P(0)$, entonces dado $P(n)$ se cumple $P(succ(n))$, que es justo el principio de inducción usual. Probemos usando este principio de inducción la conmutatividad de la suma ($add :\equiv +$) en los naturales. Necesitamos encontrar

$$conm : \prod_{i,j:\mathbb{N}} (i + j) = (j + i)$$

por tanto es suficiente probar

$$conm_0 : \prod_{j:\mathbb{N}} (0 + j) = (j + 0)$$

y

$$conm_s : \prod_{i:\mathbb{N}} \left(\Big(\prod_{j:\mathbb{N}} (i + j) = (j + i)\Big) \longrightarrow \prod_{j:\mathbb{N}} succ(i) + j = j + succ(i) \right)$$

Para probar $conm_0$, basta notar: $0 + n \equiv n \equiv \underbrace{succ(succ(\ldots succ(0)))}_{n-veces} \equiv n + 0$. Ahora $conm_s$ se prueba usando un hecho que nos da la definición de $succ$, de que

$succ(n) + m \equiv succ(n + m) \equiv n + succ(m)$. Así tenemos

$$succ(i) + j \equiv succ(i + j) \underbrace{\equiv}_{*} succ(j + i) \equiv j + succ(i)$$

donde (*) marca la hipótesis de inducción. Entonces si justificamos que el hecho de que $a = b$ implica $succ(a) = succ(b)$ usando $refl_{\mathbb{N}}$, entonces $i + j = j + i$ justifica $succ(i + j) = succ(j + i)$. Por tanto $succ(i) + j \equiv j + succ(i)$ como queríamos demostrar. Introduciendo todo esto en $ind_{\mathbb{N}}$ tenemos la prueba que queríamos. Vale la pena notar que en la sintaxis tipada de $conm_0$ y $conm_n$, si sustituimos el símbolo $\prod$ por "para todo", podemos leer la proposición como

> *Si para todo j natural se cumple que $0+j = j+0$. Y si para todos $i, j : \mathbb{N}$ sucede que $i + j = j + i$ implica $succ(i) + j = j + succ(i)$, entonces la propocisión es cierta para todos $i, j : \mathbb{N}$.*

2.9. Tipos identidad

Como se hemos mencionado ya, en la teoría de tipos cada "porción" de la teoría es un tipo. Lo que significa que sentencias lógicas, en particular las proposiciones, son un tipo. Por ejemplo la proposición de que dos elementos del mismo tipo $a, b : A$ son iguales, debe ser un tipo. Estos tipos son conocidos como **tipos identidad** o **tipos igualdad**. Podemos escribir

$$Id_A : A \longrightarrow A \longrightarrow \mathcal{U}$$

de manera que $Id_A(a, b) : \mathcal{U}$ sea el tipo que representa la proposición de igualdad entre a y b. Este tipo, también es denotado en general por el símbolo $a =_A b$. La existencia de un elemento en $a =_A b$ es una prueba (o un testigo) de que a y b son proposicionalmente iguales. Lo cual es distinto al hecho de que en efecto sean iguales por una consecuencia de operación o del carácter constructivo de la teoría ($\equiv$). En general el tipo $a = b$ puede ser pensado de manera un poco mas rica que en el sentido tradicional. Es decir, puede existir mas de un testigo en $a = b$, lo que significaría que hay mas de una prueba de este hecho. Esto tiene un significado equivalente al de identificar puntos bajo alguna relación en un espacio topológico. De la misma forma que pueden haber varias trayectorias que unan dos puntos pueden existir varios elementos en $a = b$ que "identifiquen" a a y b. Dicho lo anterior podemos

establecer como regla de formación que dados $a, b : \mathcal{U}$, existe el tipo $a =_A b$. La regla de introducción de este tipo es una función dependiente

$$refl : \prod_{a:A} (a =_A a)$$

esta es llamada reflexividad. Digamos que establece el hecho de que cada elemento esta relacionado consigo mismo por esta relación de igualdad. En nuestro esceneario de trayectorias esto correspondería a una trayectoria constante en el punto a. Decimos ademas que si $a \equiv b$ entonces $refl_A : a =_A b$. El principio de inducción para familias de tipos de identidades es un concepto bastante engorroso, aunque fundamental en la teoría homotópica de tipos. El siguiente principio establece el fundamento de la definición mas no la definición en si misma

Indistinción de identicos: Para cada familia

$$C : A \longrightarrow \mathcal{U}$$

hay una función

$$f : \prod_{(x,y:A)} \prod_{(p:x=_A y)} C(x) \longrightarrow C(y)$$

tal que $f(x, x, refl_x) :\equiv Id_{C(x)}$.

El principio anterior establece que las familias de tipos en este caso $C : \mathcal{U}$, respetan la igualdad entre los elementos del tipo en el sentido de que si dos elementos son iguales en A existe una función entre los tipos generados por ellos en la estructura dependiente. En particular si uno elige estos dos elementos como el mismo (i.e toma $refl_x : x =_A x$), entonces la función que resulta es $Id_{C(x)}$. Es importante notar que el elemento p no es tomado en cuenta en la naturaleza de la relación $C(x) \longrightarrow C(y)$, es solo un término necesario para la existencia de la misma. La inducción se basa en lo anterior pero no se limita a tipos dependientes en los que está permitida la sustitucón. Si no que se extiende a familias $F : \prod_{x,y:A} (x =_A y) \longrightarrow \mathcal{U}$.

2.9.1. Inducción de trayectorias

El principio de la inducción de trayectorias se establece de la siguiente forma

Inducción de trayectorias: Dada una familia

$$F : \prod_{x,y:A} (x =_A y) \longrightarrow \mathcal{U}$$

y una función

$$f : \prod_{x:A} F(x, x, refl_x),$$

existe una función

$$g : \prod_{(x,y:A)} \prod_{(p:x=_A y)} F(x, y, p)$$

que cumple

$$g(x, x, refl_x) := f(x).$$

En este caso F depende de p, lo que significa que la existencia de un testigo en $x =_A y$ define como es la trayectoria entre el tipo $F(x, x, refl_x)$ y $F(x, y, p)$. De hecho sugiere un poco mas, pues sugiere el hecho de que si F no es constante existen mas de una prueba de este hecho. La sintaxis anterior sugiere que existe una trayectoria desde cada elemento $F(x, y, p)$ a el punto origen $F(x, x, relf_x)$, es una estructura similar a las trayectorias basadas en un punto de la teoría de homotopías. Poniendo todos los elementos del principio de infucción anterior en una sola fórmula tenemos

$$ind_{=_A} :\equiv \prod_{(F:\prod_{x,y:A}(x=_A y)\longrightarrow \mathcal{U})} \left(\prod_{(x:A)} F(x, x, refl_x) \right) \longrightarrow \prod_{(x,y:A)} \prod_{(p:x=_A y)} F(x, y, p)$$

definido por la igualdad

$$ind_{=_A}(F, f, x, x, refl_x) :\equiv f(x).$$

La intuicón antes mencionada de que la familia anterior representa de alguna manera una serie de caminos del elemento "básico" a cada una de las conbinaciones de elementos $x, y : A$ se evidencia en el principio de inducción de caminos basados

Inducción de caminos basados: Fijemos un elemento $a : A$ y supongamos que dada una familia

$$F : \prod_{x:A} (a =_A x) \longrightarrow \mathcal{U}$$

y un elemento

$$c : F(a, refl_a),$$

tenemos que para cada

$$f : \prod_{(x:A)} \prod_{(p:a=_A x)} F(x, p),$$

se cumple

$$f(a, refl_a) :\equiv c.$$

El principio anterior de alguna manera establece que dado que $F(x, p)$ es una familia dependiente de $x : A$ y un testigo de $p : a =_A x$, entonces para definir una función en dada familia solo tenemos que fijarnos en el elemento base $a : A$. Es decir, el valor $f(a, refl_a)$ define a la función f. El sentido homotópico de estos principios son mas evidentes en el proximo capítulo. Escribiendo este segundo principio en una fórmula inductiva tenemos

$$ind'_{=_A} :\equiv \prod_{a:A} \left(\prod_{(F:\prod_{x:A}(a=_A x)\to\mathcal{U})} C(a, refl_a) \longrightarrow \prod_{(x:A)} \prod_{(p:a=_A x)} F(x, p) \right)$$

definido por

$$ind'_{=_A}(a, F, f, a, refl_a) :\equiv c.$$

Todo lo escrito anteriormente puede ser un poco confuso en términos de igualdad de elementos. Por ejemplo, el principio de inducción de los numeros naturales establece que todos los números naturales son 0 y elementos de la forma $succ(n)$. El principio de inducción para tipos de indentidad establece que todas las identidades son iguales a alguna $refl_x$ con $x : A$. De alguna manera es confuso establecer que el tipo $x =_A y$ puede tener varios elementos, y luego decir que todos los elementos aqui son equivalentes a $refl_x$. El sentido que tiene esto es establecer que las familias de tipos $x =_A y$ cuando x e y varían sobre A están definidas inductivamente por $refl_x$. Digamos que es una relación similar a la que existe entre los naturales y el elemento $0 : \mathbb{N}$. Por lo anterior, se hace obvio que si uno fija el elemento $a : A$, cualquier familia definida sobre $a =_A x$ cuando x varía sobre A está definida inductivamente por $refl_a$. Esto es equivalente a decir, como ya se mencionó a que las trayectorias basadas en $a : A$ están de alguna manera definidas por su “equivalencia“ a través de $refl_a$.

2.9.2. Equivalencia de ind e ind'

Veamos que los dos principios de inducción anteriores son equivalentes. Una implicación es bastante sencilla, la implicación donde podemos deducir $ind_{=_A}$ de $ind'_{=_A}$. Dadas las hipótesis de $ind_{=_A}$

$$C : \prod_{x,y:A} (x =_A y) \longrightarrow \mathcal{U} \ ,$$

$$f : \prod_{x:A} C(x, x, refl_x).$$

es posible fijar un elemento $a : A$ y lo anterior se transforma en

$$C' : \prod_{x:A} (a =_A x) \longrightarrow \mathcal{U} \ ,$$

$$C' :\equiv C(a),$$

$$c' : C(a, refl_a),$$

$$c' :\equiv c(a).$$

Entonces es fácil ver en la construcción de la inducción de caminos basados, que C' y c' cumplen con las premisas necesarias, así es posible como hicimos antes, tomar la función

$$g : \prod_{x:A} \prod_{(p:a=_A x)} C'(x, p)$$

que se define mediante la ecuación

$$g(x, refl_x) :\equiv c'.$$

Entonces tomando una función

$$f : \prod_{(x,y:A)} \prod_{(p:x=_A y)} C(x, y, p)$$

y recordando el hecho de haber fijado $a : A$ anteriormente, tenemos

$$f(x, x, refl_x) :\equiv g(x, refl_x) :\equiv c' :\equiv c(x).$$

La otra dirección es un poco mas complicada, pues no queda claro como podemos generar una instancia de $ind'_{=_A}$ a partir de $ind_{=_A}$. Pero podemos instanciar todas las

posibles representaciones de $ind'_{=_A}$ a través de $ind_{=_A}$. Definimos

$$D : \prod_{x,y:A} (x =_A y) \longrightarrow \mathcal{U} \ ,$$

$$D(x, y, p) :\equiv \prod_{C:\prod_{z:A}(x=_A z)\longrightarrow\mathcal{U}} C(x, refl_x) \longrightarrow C(y, p).$$

Entonces podemos construir la función

$$d : \prod_{x:A} D(x, x, refl_x),$$

$$d :\equiv \lambda x.\lambda C.\lambda(c : C(x, refl_x)).c$$

y por tanto, usando inducción de trayectorias tenemos

$$f : \prod_{(x,y:A)(p:x=_A y)} D(x, y, p)$$

con $f(x, x, refl_x) :\equiv d(x)$. Usando la definición de D podemos deducir la representación del tipo de f

$$f : \prod_{(x,y:A)(p:x=_A y)(C:\prod_{z:A}(x=_A z)\longrightarrow\mathcal{U})} C(x, refl_x) \longrightarrow C(y, p).$$

Y ahora dados $x : A$ y $p : a =_A x$, podemos deducir

$$f(a, x, p, C, c) : C(x, p).$$

Vale la pena notar que esta es justo la fórmula de la igualdad por la definición.

Capítulo 3

Teoría Homotópica de Tipos

Recapitulando el principio de inducción de los tipos de indentidad, tenemos lo siguiente si

- Para cada $x, y : A$ y cada $p : x =_A y$, tenemos la familia $D(x, y, p)$ y
- Para cada $a : A$ tenemos un elemento $d(a) : D(a, a, relf_a)$

entonces se cumple que

- existe un elemento $ind_{=_A}(D, d, x, y, p) : D(x, y, p)$ para cada $x, y : A$ y cada $p : x =_A y$ que cumple $ind_{=_A}(D, d, a, a, refl_a) \equiv d(a)$.

Esto se puede reescribir en sintaxis tipada de la siguiente forma. Dadas funciones dependientes

$$D : \prod_{(x,y:A)} (x = y) \longrightarrow \mathcal{U}$$

$$d : \prod_{(a:A)} D(a, a, refl_a)$$

hay una función dependiente

$$ind_{=_A}(D, d) : \prod_{(x,y:A)} \prod_{(p:x=_A y)} D(x, y, p)$$

tal que

$$ind_{=_A}(D, d, a, a, refl_a) \equiv d(a).$$

para cada $a : A$. Informalmente lo anterior enuncia el hecho de que cuando queramos construir un testigo (i.e encontrar una prueba) en una familia que dependa de un elemento $p : x =_A y$. Basta con considerar el caso donde x e y son iguales y p es $refl_x : x =_A x$. Esta "reducción" al caso reflexivo es similar a tomar el caso base y el paso de inducción en una prueba hecha en los números naturales.

3.1. Tipos como grupoides superiores

En esta sección estamblecemos en términos de la teoría tipada las propiedades y operaciones usuales entre trayectorias (i.e inversión, composición y neutralidad). Para esto estableceremos los resultados como "pruebas" de lemas y teoremas. Recordemos que en la teoría de tipos las proposiciones son tipos. Y los lemas y teoremas son tipos habitados, es decir, tipos en los cuales hemos encontrado un testigo, por tanto una prueba. Entonces cualquier teorema o lema debe ser "escrito" como un tipo, y debemos "encontrar" o "escribir" un testigo de dicho tipo para tener una prueba del mismo.

Lema 3.1.1. *Para cada tipo A y cada elementos $x, y : A$, existe una función*

$$(x = y) \longrightarrow (y = x)$$

denotada por $p \longrightarrow p^{-1}$. De manera que $refl_x^{-1} \equiv refl_x$ para cada $x : A$. Llamamos a p^{-1} el **inverso** *de p.*

Demostración: La demsotración del teorema consiste en construir un testigo en este tipo, es decir, crear un elemento

$$f : \prod_{(A:\mathcal{U})} \prod_{(x,y:A)} (x = y) \longrightarrow (y = x)$$

En realidad hay dos formas de hacer esto. De una manera muy formal derivando cada elemento de cada tipo involucrado en la demostración, o "escribiendo" los argumentos en lenguaje natural de manera que este contenga una traducción de dichos tipos involucrados. Para mas claridad sobre este hecho escribimos las dos formas.

Supongamos $A : \mathcal{U}$, y tomemos $D : \prod_{(x,y:A)} (x = y) \longrightarrow \mathcal{U}$ como la familia $D(x, y, p) :\equiv (y = x)$. Es decir D es una función dependiente que toma $x, y : A$, $p : x =_A y$ y

asigna el tipo $y = x$. Siguiendo el esquema inductivo del tipo identidad tenemos entonces un elemento

$$d :\equiv \lambda x.refl_x : \prod_{x:A} D(x, x, refl_x)$$

pues de hecho $refl_x : x =_A x$. Así la inducción nos da un elemento $ind_{=_A}(D, d, x, y, p) : (y = x)$ para cada $p : (x = y)$. Solo basta entonces definir $p^{-1} :\equiv ind_{=_A}(D, d, x, y, p)$, o de manera mas general $(-)^{-1} :\equiv \lambda p.ind_{=_A}(D, d, x, y, p)$. Según la regla de conversión

$$refl_x^{-1} \equiv ind_{=_A}(D, d, x, x, refl_x) \equiv refl_x$$

como se quería probar.

Como ya se dijo una segunda prueba, la coloquial, es mucho mas conveniente y se lleva a cabo a través de lenguaje natural. Queremos construir, dados elementos $x, y : A$ y $p : x = y$, un elemento $p^{-1} : y = x$. Según la regla de inducción de el tipo identidad, es suficiente tomar en cuenta el caso donde x e y son el mismo elemento y p es $refl_x : x = x$. Pero en tal caso, ambos tipos $x = y$, $y = x$ son el mismo tipo $x = x$. Entonces p y p^{-1}, ambos son $refl_x$. Entonces en la reducción reflexiva simplemente podemos definir $refl_x^{-1}$ tal que sea $refl_x$. De la misma forma que anteriormente, dado todo el razonamiento la regla de conversión permite establecer $refl_x^{-1} \equiv refl_x$.□

Podemos pasar a probar ahora la propiedad de transitividad de la igualdad que en lenguaje homotópico se manifiesta como “concatenación de trayectorias“.

Lema 3.1.2. *Para cada tipo $A : \mathcal{U}$ y cada elementos $x, y, z : A$ existe una función*

$$(x = y) \longrightarrow (y = z) \longrightarrow (x = z)$$

que se escribe como $p \longrightarrow q \longrightarrow p * q$, tal que $refl_x * refl_x \equiv refl_x$ para todo $x : A$. El elemento $p * q$ se conoce como composición de p y q.

Demostración: La demostración la daremos en dos estilos como hicimos con la del lema anterior. Primero tomemos $D : \prod_{(x,y:A)} (x = y) \longrightarrow \mathcal{U}$ una familia definida por

$$D(x, y, p) : \prod_{(z:A)} \prod_{(q:y=z)} (x = z).$$

Notemos el hecho de que $D(x, x, refl_x) \equiv \prod_{(z:A)} \prod_{(q:x=z)} (x = z)$. Es decir que para encontrar una d que aplicar en nuestro principio de inducción y derivar el inductor

de la igualdad $(x = z)$ necesitamos

$$d : \prod_{(z:A)} D(x, x, refl_x),$$

que es lo mismo que decir (al sustituir el valor definido de D)

$$d : \prod_{(x,z:A)} \prod_{(q:x=z)} (x = z).$$

Para esto, tomemos $E(x, z, q) :\equiv (x = z)$. Esto hace que $E(x, x, refl_x) \equiv (x = x)$, es decir

$$refl_x : E(x, x, refl_x)$$

por lo tanto el tipo $E(x, x, refl_x)$ esta habitado. Asi que tomando $e(x) :\equiv refl_x$, podemos usar el principio de inducción sobre el par (E, e), entonces tenemos una función

$$d : \prod_{(x,z:A)} \prod_{(q:x=z)} E(x, z, q)$$

es decir, tomamos $d :\equiv ind_{=_A}(E, e)$. Y como $E(x, z, q) :\equiv (x = z)$, tenemos

$$d : \prod_{(x,z:A)} \prod_{(q:x=z)} (x = z).$$

Aplicando entonces el principio de inducción sobre el par (D, d), tenemos

$$\begin{aligned} ind_{=_A}(D, d) : & \prod_{(x,y:A)} \prod_{(p:x=y)} \prod_{(z:A)} \prod_{(q:y=z)} (x = z) \\ & : \prod_{(x,y,z:A)} \prod_{(p:x=y)} \prod_{(q:y=z)} (x = z) \\ & \equiv \prod_{(x,y,z:A)} (x = y) \longrightarrow (y = z) \longrightarrow (x = z) \end{aligned}$$

evaluando el principio de conversión de ambas inducciones nos da $refl_x * refl_x \equiv refl_x$.

Si queremos ver la demostración de manera coloquial basta con invocar el principio de inducción dos veces como en la demostración previa. Dado que queremos construir, para todos $x, y, z : A$ y cada $p : x = y$, $q : y = z$ una prueba de que $(x = z)$. Entonces basta invocar el principio de inducción sobre p y reducir al caso donde x e y son el mismo elemento. Entonces p es $refl_x$. Cuando esto sucede el tipo $(y = z)$ del cual q es un elemento, se convierte en el tipo $(x = z)$, que invocando el principio de inducción

sobre q tenemos que z es x y q es $refl_x$. En dado caso tenemos $refl_x : (x = x)$. Es decir que tenemos un representante en $(x = z)$.□

Los dos lemas anteriores prueban la "simetría" y la "transitividad" de las trayectorias definidas por igualdades. Estas propiedades hacen emerger dos operaciones, díganse inversión y composición. Ahora hay que verificar que estas operaciones nuevas se comportan bien. La idea de la visión homotópica en la teoría de tipos, como ya se ha dicho, es crear y estudiar estructuras de orden superior. En este caso la inversión es una operación en el tipo de las igualdades de otro tipo. Digamos que es una operación de primer orden. El caso para niveles superiores es similar pero delicado. Supongamos por ejemplo que tenemos $p : x = y$ y $q : x = y$, entonces $p = q$ es en realidad $p =_{x=y} q$, es decir una igualdad (o trayectoria) de segundo orden (entre trayectorias), lo que se conoce como homotopía. De forma que hay que verificar de manera abstracta (independientemente del orden) que estas son compatibles con las estructura tipada en general. Como por ejemplo, entender que significa la asociatividad de la composición de trayectorías, o como se comporta la concatenación con la inversión.

Lema 3.1.3. *Supongamos que tenemos $A : \mathcal{U}$, $x, y, z, w : A$, $p : x = y$, $q : y = z$, $r : z = w$. Entonces se cumplen las siguientes*

(i) $p = p * refl_y$ *y* $p = refl_x * p$

(ii) $p^{-1} * p = refl_y$ *y* $p * p^{-1} = refl_x$

(iii) $(p^{-1})^{-1} = p$

(iv) $(p * q) * r = p * (q * r)$

Demostración:

(i) $p = p * refl_y$ y $p = refl_x * p$:
Primero tomemos $D : \prod_{(x,y:A)} (x = y) \longrightarrow \mathcal{U}$ una familia definida por

$$D(x, y, p) :\equiv (p = p * refl_y)$$

Entonces procedemos como antes: $D(x, x, refl_x) \equiv refl_x = refl_x * refl_x$. Como $refl_x * refl_x \equiv refl_x$ por la definición de $*$, tenemos que $D(x, x, refl_x) \equiv$

$refl_x = refl_x$. Entonces hay una

$$d :\equiv \lambda x.refl_{refl_x} : \prod_{x:A} D(x, x, refl_x).$$

El principio de inducción nos asegura un elemento

$$ind_{=_A}(D, d, p) : (p = p * refl_y)$$

para cada $p : x = y$. La otra igualdad se prueba de forma similar, tomando $D(x, y, p) :\equiv (p = refl_x * p)$.

(ii) $p^{-1} * p = refl_y$ y $p * p^{-1} = refl_x$:
Aq̌i tomamos $D : \prod_{(x,y:A)} (x = y) \longrightarrow \mathcal{U}$ una familia definida por

$$D(x, y, p) :\equiv (p^{-1} * p = refl_y)$$

Entonces $D(x, x, refl_x) \equiv (refl_x^{-1} * refl_x = refl_x)$. Pero ya probamos que $refl_x^{-1} \equiv refl_x$, entonces $D(x, x, refl_x) \equiv (refl_x = refl_x)$. Entonces como en el inciso anterior tenemos

$$d :\equiv \lambda x.refl_{refl_x} : \prod_{x:A} D(x, x, refl_x).$$

El principio de inducción nos da un elemento

$$ind_{=_A}(D, d, p) : (p^{-1} * p = refl_y)$$

para cada $p : x = y$.

(iii) $(p^{-1})^{-1} = p$:
Sea la familia $D : \prod_{(x,y:A)} (x = y) \longrightarrow \mathcal{U}$ definida como

$$D(x, y, p) :\equiv ((p^{-1})^{-1} = p)$$

Entonces $D(x, x, refl_x) \equiv (refl_x^{-1} * refl_x = refl_x)$. Usando el hecho ya mencionado $refl_x^{-1} \equiv refl_x$, tenemos

$$(refl_x^{-1})^{-1} \equiv refl_x^{-1} \equiv refl_x$$

para cada $x : A$. Y de nuevo $D(x, x, refl_x) \equiv (refl_x = refl_x)$. Lo que nos da una función

$$d :\equiv \lambda x.refl_{refl_x} : \prod_{x:A} D(x, x, refl_x).$$

y un testigo $ind_{=_A}(D, d, p) : ((p^{-1})^{-1} = p)$

(iv) $(p * q) * r = p * (q * r)$:
Tomemos la familia $D : \prod_{(x,y:A)} (x = y) \to \mathcal{U}$ dada por

$$D(x, y, p) :\equiv \prod_{(z,w:A)} \prod_{(q:y=z)} \prod_{(r:z=w)} ((p * q) * r = p * (q * r))$$

Vemos que

$$D(x, x, refl_x) \equiv \prod_{(z,w:A)} \prod_{(q:x=z)} \prod_{(r:z=w)} ((refl_x * q) * r = refl_x * (q * r))$$

por tanto debemos encontrar una función d en este tipo para poder usar el pricipio de inducción sobre el par (D, d). Para esto definimos $D_1(x, z, q)$ como

$$D_1(x, z, q) :\equiv \prod_{(w:A)} \prod_{(r:z=w)} ((refl_x * q) * r = refl_x * (q * r))$$

y notamos que

$$D_1(x, x, refl_x) \equiv \prod_{(w:A)} \prod_{(r:x=w)} ((refl_x * refl_x) * r = refl_x * (refl_x * r))$$

pero para poder aplicar inducción sobre algún par (D_1, d_1) necesitamos encontrar tal $d_1 : D_1(x, x, refl_x)$. Para esto definimos D_2 tal que

$$D_2(x, w, r) :\equiv ((refl_x * refl_x) * r = refl_x * (refl_x * r))$$

y en este tipo se cumple que

$$D_2(x, x, refl_x) \equiv ((refl_x * refl_x) * refl_x = refl_x * (refl_x * refl_x)).$$

Por otro lado, sabemos que $refl_x * refl_x \equiv refl_x$. Así que $D_2(x, x, refl_x) \equiv refl_x = refl_x$. Por tanto tomando

$$d_2 :\equiv \lambda x.refl_{refl_x} : \prod_{x:A} D_2(x, x, refl_x).$$

y haciendo todo el procedimiento de regreso (i.e tomando el inductor de el paso i como d_{i-1}). Tenemos lo que queríamos probar $ind_{=_A}(D, d, p) : ((p * q) * r = p * (q * r))$.□

El lema anterior tiene varias cosas que notar. En primer lugar, la evidencia de un procedimiento bastante estándar a través del cual podemos casi algoritmizar la demostración. Haciendo uso del principio de inducción del tipo de las identidades podemos definir estructuras de identidades de orden superior tan "arriba" como queramos, todas basadas en este principio. Lo que nos permite de alguna manera determinar el nivel de detalle con el cual abordamos la demostración. Bien visto el unico detalle es escribir la familia inicial y proceder con los pasos de inducción. Estos pasos en general se realizan de izquierda a derecha en la sintaxis, solo para mantener un procedimiento estándar, ya que el orden es realmente transparente para el resultado final.

Otra cosa que es importante notar es que el objeto que emerge del concepto identidad (o igualdad), dígase por ejemplo $a = a$, con $a : A$ es totalmente trivial en la teoría de conjuntos. Sin embargo en la teoría de tipos (más aún en la teoría homotópica de tipos), es un concepto fundamental donde se construyen los objetos iniciales de la teoría de tipos de orden superior. Por ejemplo pensemos en el tipo $a = a$. Este tipo en particular contiene $refl_a$, pero puede en general contener otros objetos que evidencian dicha igualdad. Pensando en términos homotópicos estos elementos son lazos, lo que nos da la siguiente definición.

Definición 3.1.1. *Sea $A : \mathcal{U}$ un tipo y $a : A$. Se define $\Omega(A, a)$ como el tipo $a =_A a$. El punto a puede ser omitido si se intuye facilmente del contexto.*

Como los elementos de $\Omega(A)$ pueden ser compuestos (concatenados u operados con el operador $*$), entonces esto nos define una operación $\Omega A \times \Omega A \longrightarrow \Omega A$.

De manera similar podemos pensar en los lazos de lazos basados en a. Por ejemplo podemos tomar $refl_a =_{a=_A a} refl_a$, estos son elementos de un tipo denotado como $\Omega^2(A, a)$, que es en otras paralabras, el tipo de los lazos de segundo orden o 2-lazos en a. Es claro que los 2-lazos son lazos sobre los 1-lazos, y que la construcción se extiende *ad infinitum*.

Tenemos el siguiente teorema

Teorema 3.1.1 (Eckmann–Hilton)**.** *La operación de composición sobre el 2-espacio de lazos*

$$\Omega^2(A) \times \Omega^2(A) \longrightarrow \Omega^2(A)$$

es conmutativa. Esto es: $\alpha * \beta = \beta * \alpha$*, para todos* $\alpha,\beta : \Omega^2(A)$*.*

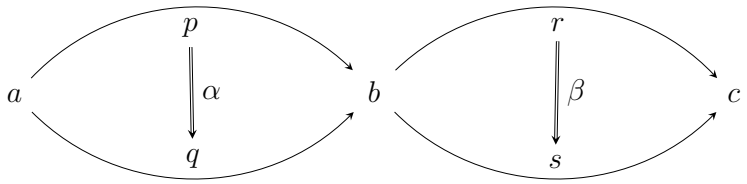
Demostración: Primero veamos que la composición en $\Omega(A)$ induce una composición

$$\star : \Omega^2(A) \times \Omega^2(A) \longrightarrow \Omega^2(A)$$

de la siguiente manera. Consideremos elementos a, b, c y caminos

$$p : a = b \qquad r : b = c$$
$$q : a = b \qquad s : b = c$$
$$\alpha : p = q \qquad \beta : r = s$$

que pueden ser representados en el siguiente diagrama



De manera que al componer $p * r, q * s : a = s$, se induce una composición

$$\alpha \star \beta : p * r = q * s.$$

Para definir la composición $\star$, definamos primero los elementos

$$\alpha *_r r : p * r = q * r$$

$$q *_l \beta : q * r = q * s$$

Estas operaciones $*_r$ y $*_l$ se conocen como **whiskering** (algo como abigotar o afibrar). Definimos $\alpha *_r r$ usando inducción de trayectorias sobre r con el procedimiento usual, tenemos que

$$\alpha *_r refl_b \equiv ru_p^{-1} * \alpha * ru_q$$

donde $ru_p : p = p * refl_b$. Que es un elemento cuya existencia esta dada por el Lema 3.1.3(i). El sentido de la equivalencia de arriba esta dada por la trayectoria

$$p * refl_b \longrightarrow p \longrightarrow q \longrightarrow q * refl_b$$

De igual forma podemos definir $q *_l \beta$ tal que $refl_b * \beta \equiv lu_r^{-1} * \beta * lu_s$, que es

$$refl_b * r \longrightarrow r \longrightarrow s \longrightarrow refl_b * s$$

Es obvio que las trayectorias $\alpha *_r r$ y $q *_l \beta$ son ensamblables, así que definimos la **composición horizontal** por

$$\alpha \star \beta : (\alpha *_r r) * (q *_l \beta).$$

Supongamos ahora que $a \equiv b \equiv c$, lo que significa que $p, q, r, s : \Omega\,(A, a)$. Supongamos ademas que $p \equiv q \equiv r \equiv s \equiv refl_a$. Entonces α,β : $refl_a = refl_a$, con concatenables en cualquier orden. Así tenemos

$$\begin{aligned}
\alpha \star \beta &\equiv (\alpha *_r refl_a) * (refl_a *_l \beta) \\
&\equiv (ru^{-1}_{refl_a} * \alpha * ru_{refl_a}) * (lu^{-1}_{refl_a} * \beta * lu_{refl_a}) \\
&\equiv (refl^{-1}_{refl_a} * \alpha * refl_{refl_a}) * (refl^{-1}_{refl_a} * \beta * refl_{refl_a}) \\
&\equiv refl^{-1}_{refl_a} * \alpha * refl_{refl_a} * refl^{-1}_{refl_a} * \beta * refl_{refl_a} \\
&\equiv \alpha * \beta
\end{aligned}$$

donde $ru_{refl_a} \equiv lu_{refl_a} \equiv refl_{refl_a}$ se da por la regla de computación de la inducción mencionada arriba.

Podemos realizar un procedimiento similar sustituyendo r por s y q por p para obtener

$$\alpha \star' \beta :\equiv (p *_l \beta) * (\alpha *_r s)$$

derivando el resultado equivalente

$$\alpha \star' \beta \equiv (refl_a *_l \beta) * (\alpha *_r refl_a) \equiv \beta * \alpha$$

pero dado que $\alpha : p = q$ y $\beta : r = s$ y $a \equiv b \equiv c$. La definición es indistinta en este caso. Por tanto se cumple $\alpha \star \beta \equiv \beta \star' \alpha$.□

Cabe la pena mencionar que de algunos hechos enunciados en el teorema anterior y del Lema 3.1.3 se deduce fácilmente que

$$\alpha *_r (p * q) \equiv (\alpha *_r p) *_r q).$$

Por otro lado, estuvimos usando en el teorema también, que nuestros lazos estaban todos "basados" en el punto $a : A$. Aunque en el teorema anterior no se uso este hecho en la demostración es importante formalizar el concepto.

Definición 3.1.2. *Un* **tipo punteado** *consiste en un tipo $A : \mathcal{U}$ y un punto $a : A$ llamado* **punto base**. *Denotamos como $\mathcal{U}_\bullet :\equiv \sum_{A:\mathcal{U}} A$ el tipo de los tipos punteados en $\mathcal{U}$.*

Definición 3.1.3. *Dado un tipo punteado (A, a), definimos el* **espacio de lazos** *sobre (A, a) como el tipo punteado*

$$\Omega(A, a) :\equiv ((a =_A a), refl_a).$$

Los elementos de este tipo son llamados **lazos** *en a.*

La definición anterior se puede extender al **n-espacio de lazos** si tomamos $n : \mathbb{N}$ y recursivamente definimos

$$\Omega^0(A, a) :\equiv (A, a)$$

$$\Omega^{n+1}(A, a) :\equiv \Omega(\Omega^n(A, a)).$$

3.2. Funciones como funtores

Como el título de la sección sugiere, veamos que las funciones tienen un comportamiento funtorial[1] sobre las trayectorias. En particular esto quiere decir que las funciones "respetan" la igualdad entre elementos de su domino y su codominio.

Lema 3.2.1. *Sea $f : A \longrightarrow B$ una función. Entonces existe*

$$ap_f : (x =_A y) \longrightarrow (f(x) =_B f(y))$$

mas aún, se cumple: $ap_f(refl_x) \equiv refl_{f(x)}$, para cada $x : A$.

[1] Ver apéndice **B.2**

Demostración: Antes de pasar a la demostración vale la pena destacar que siguiendo el estilo usual de la teoría de tipos la función ap toma su nombre de **ap**plication. Lo cual es bastante acertado ya que justamente se comporta como un operador acplicación.

La demostración sigue el esquema usual. Sea la familia $D : \prod_{x,y:A} (x = y) \longrightarrow \mathcal{U}$, definida como

$$D(x, y, p) :\equiv (f(x) = f(y))$$

tenemos entonces que $D(x, x, refl_x)$ nos da un elemento

$$d :\equiv \lambda x.refl_{f(x)} : \prod_{x:A} D(x, x, refl_x)$$

Por inducción de trayectorias existe

$$ind_{=_A}(D, d, x, y, p) : \prod_{(x,y:A)} \prod_{(p:x=y)} (f(x) = f(y))$$

que es

$$ap_f(p) :\equiv ind_{=_A}(D, d, x, y, p) : \prod_{x,y:A} (x = y) \longrightarrow (f(x) = f(y))$$

donde p es el testigo de la igualdad $(x = y)$. Al aplicar la regla de computación tenemos

$$ap_f(refl_x) \equiv ind_{=_A}(D, d, x, x, refl_x) \equiv d(x) \equiv refl_{f(x)}$$

Como se quería demostrar. □

En algunos casos (dado que el comportamiento es funtorial) se puede abusar del lenguaje e intercambiar $ap_f(p)$ por $f(p)$. Si es sabido el hecho de que p es una igualdad (i.e o una trayectoria en la homotopía), entonces el teorema anterior nos dice que f manda esta trayectoria a una en B (i.e f es continua). Entonces no es ambigua la notación $f(p)$ para trayectorias p.

Lema 3.2.2. *Si tenemos funciones $f : A \longrightarrow B$ y $g : B \longrightarrow C$, y elementos $p : x =_A y$, $q : y =_A z$. Entonces*

(i) $ap_f(p * q) = ap_f(p) * ap_f(q)$

(ii) $ap_f(p^{-1}) = ap_f(p)^{-1}$

(iii) $ap_g(ap_f(p)) = ap_{g\circ f}(p)$

(iv) $ap_{id_A}(p) = p$.

Demostración: La demostración sigue el mismo esquema de las anteriores

(i) $ap_f(p * q) = ap_f(p) * ap_f(q)$:
Queremos demostrar dadas las condiciones del lema, que $f(p*q) = f(p)*f(q)$. Para esto usemos inducción sobre p, es decir queremos un testigo de

$$f(refl_x * q) = f(refl_x) * f(q)$$

pero usando que $q \equiv refl_x * q$, $f(refl_x) \equiv refl_{f(x)}$ y $f(q) \equiv (f(x) = f(z))$. Tenemos

$$refl_{f(q)} : f(refl_x * q) = f(refl_x) * f(q)$$

(ii) $ap_f(p^{-1}) = ap_f(p)^{-1}$:
Esta propiedad es casi trivial del hecho de que $refl_x^{-1} \equiv refl_x$. Es decir, tomando inducción sobre p

$$ap_f(refl_x^{-1}) \equiv ap_f(refl_x) \equiv refl_{ap_f(x)} \equiv refl_{ap_f(x)}^{-1} \equiv ap_f(refl_x)^{-1}$$

(iii) $ap_g(ap_f(p)) = ap_{g\circ f}(p)$:
La misma propiedad de permutabilidad entre ap y $refl_x$ nos permite demostrar este inciso. Tomando inducción sobre p tenemos

$$ap_g(ap_f(refl_x)) \equiv ap_g(refl_{ap_f(x)}) \equiv refl_{ap_g(ap_f(x))} \underset{*}{\equiv} refl_{ap_{g\circ f}(x)} \equiv ap_{g\circ f}(x)$$

Donde la equivalencia * esta dada ya que $x : A$, es decir, es una composición de funciones usual.

(iv) $ap_{id_A}(p) = p$:
Dado que $p : x = y$ y $ap_f(p) \equiv (f(x) = f(y))$, vemos que $ap_{id_A}((x = y)) \equiv (id_A(x) = id_A(y)) \equiv (x = y)$. □

3.3. Familias de tipos como fibraciones

El inconveniente para establecer reglas sobre el operador ap_f en familias de tipos es que en una familia $\prod_{x:A} B(x)$, en principio los tipos $B(x)$ y $B(y)$ son distintos. Por tanto no podemos establecer igualdad alguna entre elementos en cada uno de ellos. Sin embargo la existencia de la trayectoria p nos permite establecer determinada realción.

Lema 3.3.1 (Transporte)**.** *Supongamos que F es una familia sobre A, y $p : x =_A y$. Entonces existe una función*

$$p_* : F(x) \longrightarrow F(y).$$

Demostración: La demostración es, sorpresivamente, usando el principio de inducción. Supongamos que tenemos la familia $D : \prod_{x,y:A} (x = y) \longrightarrow \mathcal{U}$, definida como

$$D(x, y, p) :\equiv F(x) \longrightarrow F(y)$$

Entonces vemos que $D(x, x, refl_x) \equiv F(x) \longrightarrow F(x)$, es decir, tomando

$$d : \lambda x.id_{F(x)} : \prod_{x:A} F(x) \longrightarrow F(x)$$

tenemos $p_* :\equiv ind_{=_A}(D, d, x, y, p) : F(x) \longrightarrow F(y)$, para cada $p : x =_A y$. □

La notación usual para el transporte es

$$transport^P(p, -) : F(x) \longrightarrow F(y)$$

donde el supra índice puede ser omitido si es claro en que familia se establece la relación. Entonces dicho en lenguaje natural, el transporte levanta un camino p a un elemento en la fibra del inicio de dicho camino, devolviendo el final del camino una vez levantado.

Desde el punto de vista homotópico lo que esta haciendo la trayectoria p es "conectar" a todos los elementos que cumplen $p : x = y$. En otras palabras, uno puede pensar en una familia $\prod_{x:A} F(x)$ como una serie de propiedades que cumplen los elementos $x : A$. De esta manera el resultado anterior se traduce en que $F(x) = F(y)$ si y solamente si tienen la misma propiedad. Esto en términos homotópicos esta relacionado con la

propiedad que tienen algunos espacios y las trayectorias en ellos de ser "levantados" a otros espacios. Es decir, si pensamos en A como el tipo base y en el tipo $F(x)$ como la **fibra** sobre $x : A$ entonces la familia $\prod_{x:A} F(x)$ representa el espacio total y el tipo $\prod_{x:A} F(x) \longrightarrow A$ es una fibración[2]. Esta propiedad también existe en teoría de tipos.

Lema 3.3.2 (Propiedad del levantamiento de trayectorias)**.** *Supogamos que tenemos* $P : A \longrightarrow \mathcal{U}$ *. Si tomamos* $u : P(x)$ *para alguna* $x : A$ *entonces para cada* $p : x = y$ *existe*

$$lift(u, p) : ((x, u) = (y, p_*(u)))$$

en $\prod_{x:A} P(x)$*, y se cumple* $pr_1(lift(u, p)) = p$*.*

Demostración: Como es de costumbre, notemos primero que dadas nuestras hipótesis, lo que necesitamos para demostrar el lema es encontrar un testigo en $((x, u) = (y, p_*(u)))$. Definamos una familia $D : \prod_{x,y:A} (x = y) \longrightarrow \mathcal{U}$ de la siguiente manera

$$D(x, y, p) :\equiv ((x, u) = (y, p_*(u))).$$

Entonces si tomamos $D(x, x, refl_x)$, en tal caso $p_* : P(x) \longrightarrow P(x)$, es decir $p_* \equiv id_{P(x)}$. Por tanto se cumple que

$$D(x, x, refl_x) \equiv ((x, u) = (x, p_*(u))) \equiv ((x, u) = (x, id_{P(x)}(u))) \equiv ((x, u) = (x, u))$$

este tipo esta habitado por $refl_{(x=u)}$. Por ende tenemos

$$lift(u, p) :\equiv ind_{=_A}(D, d, x, y, p) : ((x, u) = (y, p_*(u)))$$

como se quería demostrar. □

Vale la pena destacar que no es correcto decir que la función $P : A \longrightarrow \mathcal{U}$ es una fibración (como sucede en la homología). el espacio total en este caso es $\prod_{x:A} P(x)$.

A veces podremos referirnos a las **fibras** o secciónes de la fibración $f : \prod_{x:A} P(x)$. O cuando intentemos demostrar algo o referirnos a alguna propiedad decir que se cumple para cada elemento de la fibra, esto es, para cada $P(x)$.

Ahora podemos probar la version del Lema 3.2.1 en su versión dependiente. La idea es establecer el hecho de que dada una función $f : \prod_{x:A} P(x)$ y un camino $p : x =_A y$,

[2]Ver apéndice **A.1**

entonces al aplicar f a p obtenemos un camino que corre "sobre" p. Es decir que $p_1(f(p)) = p$. La sugerencia es usar el transporte inducido por p. Es decir, tenemos un camino $lift(u, p)$ de (x, u) a $(y, p_*(u))$ que corre sobre p. Ahora aunque $p_*(u)$ tenga el tipo adecuado no queda claro si cualquier camino $u : P(x)$ a $v : P(y)$ que corra sobre p sea único (en el sentido de las equivalencias generadas por $=_{P(y)}$). En otras palabras necesitamos probar que cualquier camino "que corra sobre p", con $u : P(x)$, $v : P(y)$ es en esencia $p_*(u) = v$.

Lema 3.3.3 (del operador aplicación dependiente)**.** *Tomemos* $f : \prod_{x:A} P(x)$*, entonces tenemos una función dependiente*

$$apd_f : \prod_{p:x=y} (p_*(f(x)) =_{P(y)} f(y)).$$

Demostración: Tomemos una familia $D : \prod_{x,y:A} (x = y) \longrightarrow \mathcal{U}$ definida por

$$D(x, y, p) :\equiv p_*(f(x)) =_{P(y)} f(y).$$

Entonces $D(x, x, refl_x) \equiv p_*(f(x)) =_{P(x)} f(x) \equiv (refl_x)_*(f(x)) = f(x) \equiv id_{P(x)}(f(x)) = f(x)$. Así

$$d :\equiv \lambda x.refl_{f(x)} : \prod_{x:A} D(x, x, refl_x)$$

y la inducción nos da $apd_f(p) : p_*(f(x)) = f(y)$ para cada $p : x = y$. □

De la misma manera que las funciónes constantes y las familias están relacionadas, los tipos ap_f y apd_f se relacionan.

Lema 3.3.4. *Sea* $P : A \longrightarrow \mathcal{U}$ *definida como* $P(x) :\equiv B$ *un tipo* $B : \mathcal{U}$ *fijo. Entonces para cualesquiera* $x, y : A$*,* $p : x = y$ *y* $b : B$*, tenemos un camino*

$$transportconst_p^B(b) : transport^P(p, b) = b.$$

Demostración: Supongamos que tenemos $a, b : B$, y la familia $D : \prod_{x,y:A} (x = y) \longrightarrow \mathcal{U}$. Que se define como

$$D(x, y, p) :\equiv (transport^P(p, b) = b).$$

Entonces $D(x, x, refl_x) \equiv (transport^P(refl_x, b) = b)$, pero por la regla de computación de la inducción del transporte, tenemos que $transport^P(refl_x, b) \equiv (\lambda x.id_{P(x)})(b) \equiv$

b, por tanto: $D(x, x, refl_x) \equiv (b = b)$. Así la inducción nos da un elemento en $\prod_{x,y:A} \prod_{p:x=y} (transport^P(p, b) = b)$, como queríamos demostrar. □

Hay que notar que si $x, y : A$, $p : x = y$ y $f : A \longrightarrow B$, entonces $transportconst^B_p(f(x))$ es un elemento equivalente a b (en un sentido homotópico que aún no establecemos formalmente). Esta función de transporte constante, que tiene sus valores en un único tipo, es iportante ya que relaciona ap_f con apd_f de la siguiente manera.

Lema 3.3.5. *Dada $f : A \longrightarrow B$ y $p : x =_A y$, tenemos*

$$apd_f(p) = transportconst^B_p(f(x)) * ap_f(p)$$

. **Demostración:** *Sea $D : \prod_{x,y:A} (x = y) \longrightarrow \mathcal{U}$ definida por*

$$D(x, y, p) :\equiv (apd_f(p) = transportconst^B_p(f(x)) * ap_f(p)).$$

Así tenemos

$$D(x, x, refl_x) \equiv (apd_f(refl_x) = transportconst^B_p(f(x)) * ap_f(p))$$

pero se cumple en este caso que $apd_f(refl_x) \equiv transportconst^B_p(f(x)) \equiv ap_f(p) \equiv refl_x$. Así tenemos

$$refl_{refl_{f(x)}} : D(x, x, refl_x)$$

y la inducción nos da un elemento en $\prod_{x,y:A} \prod_{p:x=y} D(x, y, p)$ como queríamos. □

Probemos otras propiedades de el transporte para terminar esta sección.

Lema 3.3.6. *Dado $P : A \longrightarrow \mathcal{U}$ con $p : x =_A y$, $q : y =_A z$ y $u : P(x)$, tenemos*

$$q_*(p_*(u)) = (p * q)_*(u)$$

Demostración: Por inducción sobre p tenemos que $q_*((refl_x)_*(u)) = q_*(u)$. Pero vimos que es constructivamente correcto, por la inducción de el transporte, pensar en $(refl_x)_*$ como $id_{P(x)}$. Entonces nos queda

$$q_*((refl_x)_*(u)) = q_*(u) \equiv q_*(u) = q_*(u)$$

por tanto, el tipo $q_*(p_*(u)) = (p * q)_*(u)$ está habitado como se quería demostrar. □

Lema 3.3.7. *Dados* $f : A \longrightarrow B$, $P : B \longrightarrow \mathcal{U}$ *y cualesquiera* $p : x =_A y$, $u : P(f(x))$, *tenemos*

$$transport^{P\circ f}(p, u) = transport^{P}(ap_f(p), u)$$

Demostración: El lema se puede traducir en el hecho de que la función f conserva el camino p al camino $f(p) : B$, de esta forma el transporte se puede escribir sobre el tipo $P(f(x))$. Por inducción una vez mas sobre p vemos que lo anterior se escribe como

$$transport^{P\circ f}(refl_x, u) = transport^{P}(ap_f(refl_x), u)$$

pero $ap_f(refl_x) \equiv refl_{f(x)}$, de donde

$$transport^{P}(ap_f(refl_x), u) \equiv transport^{P}(refl_{f(x)}, u).$$

Por otro lado

$$transport^{P\circ f}(refl_x, u) \equiv (P \circ f)(refl_x) =_{P(f(x))} u$$

$$transport^{P}(ap_f(refl_x), u) \equiv transport^{P}(f(refl_x), u) \equiv P(f(refl_x)) =_{P(f(x))} u$$

y tenemos lo que queríamos probar. □

Lema 3.3.8. *Para* $P, Q : A \longrightarrow \mathcal{U}$ *, familias de funciones* $f : \prod_{x:A} P(x) \longrightarrow Q(x)$ *y* $p : x =_A y$, $u : P(x)$, *tenemos*

$$transport^{Q}(p, f_x(u)) = f_y(transport^{P}(p, u))$$

Demostración: Una vez mas hacemos inducción sobre p, para esto tomamos una familia $D : \prod_{x,y:A} (x = y) \longrightarrow \mathcal{U}$ definida por

$$D(x, y, p) :\equiv transport^{Q}(p, f_x(u)) = f_y(transport^{P}(p, u)).$$

Vemos que $D(x, x, refl_x) \equiv transport^{Q}(refl_x, f_x(u)) = f_y(transport^{P}(refl_x, u))$, y necesitamos $d : D(x, x, refl_x)$. Para esto veamos quienes son los tipos

$$transport^{Q}(refl_x, f_x(u)) \equiv refl_{Q(x)} = f_x(u)$$

$$f_y(transport^{P}(refl_x, u)) \equiv f_y(refl_{P(x)} = u) \equiv refl_{f_y(P(x))} = f_y(u)$$

pero sabemos que $f_x(u) = f_y(u)$, de esta forma tenemos

$$refl_{Q(x)} * (f_x(u) = f_y(u)) * refl_{f_y(P(x))} : D(x, x, refl_x)$$

y entonces existe un testigo donde queremos

$$ind_{=_A}(D, d, x, y, p) : transport^Q(p, f_x(u)) = f_y(transport^P(p, u))$$

lo que completa la prueba. □

3.4. Homotopías y equivalencias

Entraremos ahora en una definición mas formal de homotopía y equivalencia. Con esto nos referimos sobre todo a los conceptos hasta ahora difusos de "igualdad", "equivalencia", "trayectorias" que hemos estado usando coloquialmente (sin que esto claro, intervenga en la formalidad de las pruebas).

Definición 3.4.1. *Sean $f, g : \prod_{x:A} P(x)$, dos secciones de la familia $P : A \longrightarrow \mathcal{U}$. Una* **homotopía** *entre f y g es una función dependiente definida como*

$$(f \sim g) :\equiv \prod_{x:A} (f(x) = g(x)).$$

En principio las homotopías no son lo mismo que las igualdades $(f = g)$, sin embargo de alguna manera luego serán equivalentes.

Lema 3.4.1. *Las homotopías son relaciones de equivalencia, esto es*

(i) $\prod_{f:A \longrightarrow B} (f \sim f)$

(ii) $\prod_{f,g:A \longrightarrow B} (f \sim g) \longrightarrow (g \sim f)$

(iii) $\prod_{f,g,h:A \longrightarrow B} (f \sim g) \longrightarrow (g \sim h) \longrightarrow (f \sim h)$

Demostración:

(i) $\prod_{f:A\longrightarrow B}(f \sim f)$:
Reescribiendo la relación $(f \sim f)$ tenemos

$$\prod_{f:A\longrightarrow B}\prod_{x:A}(f(x) = f(x))$$

lo cual es trivial dado que $refl_{f(x)}$ es un testigo de este tipo.

(ii) $\prod_{f,g:A\longrightarrow B}(f \sim g) \longrightarrow (g \sim f)$:
Reescribiendo de nuevo la ecuación

$$\prod_{f,g:A\longrightarrow B}\prod_{x:A}(f(x) = g(x)) \longrightarrow (g(x) = f(x))$$

lo cual esta dado por la simetría de la identidad, que se traduce en $refl_{f(x)} = refl^{-1}_{f(x)}$.

(iii) $\prod_{f,g,h:A\longrightarrow B}(f \sim g) \longrightarrow (g \sim h) \longrightarrow (f \sim h)$: De igual forma reescribiendo la ecuación tenemos

$$\prod_{f,g,h:A\longrightarrow B}\prod_{x:A}(f(x) = g(x)) \longrightarrow (g(x) = h(x)) \longrightarrow (f(x) = h(x))$$

podemos contraerla en

$$\prod_{f,g,h:A\longrightarrow B}\prod_{x:A}\prod_{p:(f=g)}(g(x) = h(x)) \longrightarrow (f(x) = h(x))$$

ahora aplicando inducción sobre p, tenemos

$$\prod_{f,g,h:A\longrightarrow B}\prod_{x:A}(f(x) = h(x)) \longrightarrow (f(x) = h(x))$$

que esta habitada por $refl_{(f(x)=h(x))}$.

Es importante destacar que las homotopías se comportan como transformaciones naturales[3] en el sentido de que si las funciones son funtores para las trayectorias, esto es, dada $f : A \longrightarrow B$ y $p : x = y$, tenemos $ap_f(p) \equiv f(p)$. Entonces tenemos lo siguiente

[3]Ver apéndice A

Lema 3.4.2. *Supongamos que $H : f \sim g$ es una homotopía entre $f, g : A \longrightarrow B$, y sea $p : x =_A y$. Entonces se cumple*

$$H(x) * g(p) = f(p) * H(y).$$

En un diagrama

$$\begin{array}{ccc} f(x) & \xrightarrow{f(p)} & f(y) \\ {\scriptstyle H(x)}\Big\| & & \Big\|{\scriptstyle H(y)} \\ g(x) & \xrightarrow[g(p)]{} & g(y) \end{array}$$

Demostración: Por inducción podemos asumir que p es $refl_x$. Teniendo

$$H(x) * refl_{g(x)} = refl_{f(x)} * H(x)$$

y ambos lados de la igualdad son $H(x)$. □

Corolario 3.4.1. *Sea $H : f \sim id_A$ una homotopía con $f : A \longrightarrow A$. Entonces para cualquier $x : A$ tenemos*

$$H(f(x)) = f(H(x))$$

donde $f(H(x))$ denota $ap_f(H(x))$.

Demostración: Por la naturalidad de H tenemos

$$f(H(x)) * H(x) = H(f(x)) * H(x)$$

o en un diagrama

$$\begin{array}{ccc} f(f(x)) & \xrightarrow{f(H(x))} & f(x) \\ {\scriptstyle H(f(x))}\Big\| & & \Big\|{\scriptstyle H(x)} \\ f(x) & \xrightarrow[H(x)]{} & x \end{array}$$

Es decir que si hacemos un whiskering con $H(x)^{-1}$, tenemos

$$f(H(x)) = f(H(x)) * H(x) * H(x)^{-1} = H(f(x)) * H(x) * H(x)^{-1} = H(f(x))$$

como se deseaba. □

Haciendo un paréntesis podemos hablar de un hecho que no hemos tocado hasta el momento. En teoría de tipos se dice que $f : A \longrightarrow B$ es un **isomorfismo** si existe una $g : B \longrightarrow A$ tal que $f \circ g$ y $g \circ f$ sean punto a punto iguales a id_A e id_B respectivamente. Esta es una definición muy distinta al concepto de isomorfismo clásico, sin embargo es casi las misma que equivalencia homotópica.

Definición 3.4.2. *Para una funcion $f : A \longrightarrow B$, una* **cuasi-inversa** *es una tripleta (g, α, β), que consiste en una función $g : B \longrightarrow A$. Y las homotopías $\alpha : f \circ g \sim id_B$ y $\beta : g \circ f \sim id_A$. En término de teoria de tipos decimos que el tipo*

$$\sum_{g:B \longrightarrow A} ((f \circ g \sim id_B) \times (g \circ f \sim id_A))$$

es una cuasi-inversa de f, y se denota por $qinv(f)$.

Por ejemplo, la funcion $id_A : A \longrightarrow A$ tiene cuasi-inversa dada por la id_A misma y las homotopías correspondientes son $\alpha(y) :\equiv refl_y$ y $\beta(x) :\equiv refl_x$.

Otro ejemplo lo podemos dar tomando $p : x =_A y$ y $P : A \longrightarrow \mathcal{U}$, y la función

$$transport^P(p, -) : P(x) \longrightarrow P(y)$$

cuya cuasi-inversa es $transport^P(p^{-1}, -)$.

En general en teoría de tipos nunca se habla de isomorfismo o biyección, solo en casos donde los tipos se comportan como grupos. Sin embargo se puede dar una definición sobre "equivalencia".

Definición 3.4.3. *Sea el operador $isequiv(f)$ con las siguientes propiedades*

(i) *Para cada $f : A \longrightarrow B$ hay una función $qinv(f) \longrightarrow isequiv(f)$.*

(ii) *Similarmente tenemos para cada f una función $isequiv(f) \longrightarrow qinv(f)$. Lo que hace que ambas sean equivalentes.*

(iii) *Para cualesquiera dos habitantes $e_1, e_2 : isequiv(f)$ tenemos que $e_1 = e_2$.*

El operador se define como

$$isequiv(f) :\equiv (\sum_{g:B\longrightarrow A} \big(f \circ g \sim id_B\big)) \times (\sum_{f:A\longrightarrow B} \big(g \circ f \sim id_A\big)).$$

Lema 3.4.3. *El operador definido en la Definición 3.4.3 cumple (i), (ii).*

Demostración:

La demostración de *(i)* es bastante directa, tomando $(g, \alpha, g, \beta) : qinv(f)$ y escribiendo $(g, \alpha, g, \beta) : isequiv(f)$.

Para demostrar *(ii)*, supongamos que tenemos $(g, \alpha, h, \beta) : isequiv(f)$, entonces escribimos

$$g \overset{\beta}{\sim} h \circ f \circ g \overset{\alpha}{\sim} h$$

Es decir, tomamos $\gamma = \beta * \alpha : g \sim h$ y definimos $\beta' = \gamma \circ \beta : g \circ f \sim id_A$. Así $(g, \alpha, \beta') : qinv(f)$. □

El inciso *(iii)* de la definición, es algo mas complicado, pero podremos probarlo mas adelante.

Resumiendo un poco, podemos decir que una equivalencia $(A \simeq B)$ es un par de una función $f : A \longrightarrow B$ y un habitante de $isequiv(f)$. Es decir, en sintáxis tipada,

$$(A \simeq B) :\equiv \sum_{f:A\longrightarrow B} isequiv(f).$$

Para terminar esta sección, tenemos el siguiente lema.

Lema 3.4.4. *La equivalencia entre tipos es una relación de equivalencia en $\mathcal{U}$. Es decir, se cumplen*

(i) *Para cada A, la identidad id_A es una equivalencia, esto es $A \simeq A$.*

(ii) *Para cada $f : A \simeq B$, tenemos un equivalencia $f^{-1} : B \simeq A$.*

(iii) *Para cada $f : A \simeq B$ y $g : B \simeq C$, tenemos $g \circ f : A \simeq C$.*

Demostración: La identidad id_A es obviamente su inversa, asi que es una equivalencia.

Por otro lado, si $f : A \longrightarrow B$ es una equivalencia, entonces tiene una cuasi-inversa, digamos $f^{-1} : B \longrightarrow A$. Por tanto f también es cuasi-inversa de f^{-1}. Así f^{-1} es una equivalencia $B \longrightarrow A$.

Finalmente dadas $f : A \simeq B$ y $g : B \simeq C$ con cuasi-inversas f^{-1} y g^{-1}, entonces para todo $a : A$ tenemos

$$(f^{-1}g^{-1}gf)(a) = (f^{-1}f)(a) = a$$

y para todo $c : C$ tenemos

$$(gf^{-1}fg^{-1})(c) = (gg^{-1})(c) = c$$

por tanto $f^{-1} \circ g^{-1}$ es cuasi-inversa para $g \circ f$. Por tanto $g \circ f$ es equivalencia.

3.5. Estructura de grupoides superiores de formación de tipos

El la teoría de tipos, un **axioma** es un elemento que de forma "atómica" se establece como habitante de un tipo específico. Esto quiere decir que solo esta definido y gobernado por las reglas definidas en dicho tipo, distinto a lo que sucede cuando es construído a través de otros. Una vez que hemos logrado dar una definición formal de equivalencia, dedicamos esta sección a establecer de manera mas explícita y específica como se comportan las equivalencias entre elementos (a través de las identidades). O desde un punto de vista homotópico, a entender como se manifiestan las propiedades homotópicas en los tipos de manera explícita, una vez que introducimos el concepto de trayectorías y levantamientos de estas. Sucede que existen casos como los $\prod$-tipos donde este proceso es imposible usando la teoría tipada que intridujimos en el Capítulo 1 (aunque existen otras en las cuales si es posible **??**). Sin embargo sortearemos esta incapacidad introduciendo axiomas que nos permitan continuar adelante con la teoría presente en este texto.

3.5.1. Productos cartesianos

Supongamos que tenemos tipos A y B. Consideremos el producto cartesiano ya definido $A \times B$. Para cualesquiera dos elementos $x, y : A \times B$ y un camino $p : x =_{A\times B} y$ entre ellos, podemos extraer

$$pr_1(p) : pr_1(x) =_A pr_1(y)$$

$$pr_2(p) : pr_2(x) =_B pr_2(y)$$

por la funtorialidad de ap_{pr_1} y ap_{pr_2}. Esto nos da una función

$$(x =_{A\times B} y) \longrightarrow (pr_1(x) =_A pr_1(y)) \times (pr_2(x) =_B pr_2(y))$$

El sentido lógico de la sentencia de arriba es que dos pares son iguales si sus componentes son las mismas. Desde el punto de vista categórico esto quiere decir que los morfismos en un grupoide producto son pares de morfismos, i.e la flecha de arriba se puede escribir como (p_1, pr_2). Homotópicamente hablando esto refleja la intuición común para los topólogos de que los caminos en un espacio producto, son productos de caminos.

Teorema 3.5.1. *La aplicación*

$$(x =_{A\times B} y) \longrightarrow (pr_1(x) =_A pr_1(y)) \times (pr_2(x) =_B pr_2(y)) \tag{3.1}$$

es una equivalencia.

Demostración: Para construir la equivalencia, necesitamos una función en la otra dirección, esto es

$$(pr_1(x) =_A pr_1(y)) \times (pr_2(x) =_B pr_2(y)) \longrightarrow (x =_{A\times B} y) \tag{3.2}$$

Primero, por la construcción de los tipos cartesianos podemos pensar en $x \equiv (a, b)$ y $y \equiv (a', b')$. En este caso la función de arriba se escribe como

$$(a =_A a') \times (b =_B b') \longrightarrow ((a, b) =_{A\times B} (a', b')).$$

Ahora por inducción sobre los dominios A y B podemos asumir que los caminos $p : a =_A b$ y $q : a' =_B b'$ están dados. Y haciendo inducción sobre p y q tenemos

$$p \equiv refl_a, a \equiv a'$$

$$q \equiv refl_b, b \equiv b'$$

De lo que se deduce por la construcción que $(a, b) \equiv (a', b')$. Es decir que la relación $(a, b) =_{A\times B} (a', b')$ es también reflexiva.

Solo queda ver que las aplicaciones (2.1) y (2.2) son cuasi-inversas.

En la primera dirección (2.1)⇒(2.2), tomemos $r : x =_{A\times B} y$. Haciendo inducción de caminos como antes tenemos que $x \equiv y$ y r es reflexiva. Como ap_{pr_1} y ap_{pr_2} pueden se definen de forma inductiva, al tomar $r \equiv refl_x$, tenemos que $ap_{(pr_1,pr_2)}(refl_x) \equiv (refl_{pr_1 x}, refl_{pr_2 x})$. Entonces podemos asumir $x \equiv (a, b)$, así que lo anterior es $(refl_a, refl_b)$. Pero la ecuación (2.2) justamente recibe esto como input y lo envía dado el caso reducido actual es $refl_{(a,b)} \equiv refl_x$, que es $refl_x = r$.

En la otra dirección (2.2)⇒(2.1) hay involucrados tres niveles de inducción también. Comenzamos con un camino $s : (pr_1(x) =_A pr_1(y)) \times (pr_2(x) =_B pr_2(y))$, que haciendo inducción sobre x e y (en el producto cartesiano), podemos escribir como $x \equiv (a, b)$ y $y \equiv (a', b')$. Luego hacemos inducción sobre $s : (a =_A a') \times (b =_B b')$, para tener que s puede escribirse como (p, q), de esta forma terminamos haciendo doble inducción sobre p y q, para tener $p \equiv refl_a$, $q \equiv refl_b$ en donde usando (2.1) nos dice $(refl_a, refl_b) \equiv (p, q) \equiv s$. □ En particular, lo anterior nos dice que (2.1) tiene una inversa que denotaremos

$$pair^{=} :\equiv (pr_1(x) =_A pr_1(y)) \times (pr_2(x) =_B pr_2(y)) \longrightarrow (x =_{A\times B} y)$$

De alguna manera el teorema anterior nos da una forma de construir la igualdad entre pares ordenados y el elemento formado por sus coordenadas, es decir $z = (pr_1(z), pr_2(z))$. Ademas nos permite establecer reglas de descomposición de caminos por coordenadas. Lo que en el sentido homotópico correspondería a encontrar el camino por cada coordenada, o ver un camino como un par ordenado de caminos. Esto es

$$ap_{pr_1}(pair^{=}(p, q)) = p,$$

$$ap_{pr_2}(pair^{=}(p, q)) = q.$$

Lo que de alguna forma tambien establece la unicidad en el sentido de que si $r : x =_{A\times B} y$, entonces $pair^{=}(pr_1(r), pr_2(r)) = r$.

También podemos caracterizar la reflexividad, la inversión y la compsición de los elementos $z : A \times B$ usando $pair^{=}$

- $refl_{z:A\times B} = pair^{=}(refl_{pr_1(z)}, refl_{pr_2(z)})$, se sigue de hacer inducción sobre z y escribir $z = (pr_1(z), pr_2(z))$.
- $p^{-1} = pair^{=}(pr_1(p)^{-1}, pr_2(p)^{-1})$, se sigue de hacer inducción sobre $p \equiv refl_x$.
- $p*q = pair^{=}(pr_1(p)*pr_1(q), pr_2(p)*pr_2(q))$, se sigue de hacer doble inducción sobre $p \equiv refl_x \equiv q$.

En el párrafo anterior usamos indiscriminadamente (y por comodidad) $pr_i(x)$ pudiendo ser esto incorrecto dado que $ap_{pr_i}(x)$ es la aplicación correcta a usar. Nos permitimos esto por el caso, ya que es indistinto el uso aquí. Pero en general las dos aplicaciónes no coinciden.

Finalmente consideremos los caminos en familias de productos definidas punto a punto. Esto es, dadas funciones $A \times B : Z \longrightarrow \mathcal{U}$, escribimos $A \times B : Z \longrightarrow \mathcal{U}$ (abusando un poco del lenguaje), para la familia definida como $(A \times B)(z) :\equiv A(z) \times B(z)$. Dado un camino $p : z =_Z w$ y $x : A(z) \times B(z)$, podemos "transportar" x sobre un levantamiento de p para obtener el punto final $A(w) \times B(w)$.

Teorema 3.5.2. *Usando las hipótesis anteriores tenemos*

$$transport^{A\times B}(p, x) =_{(A(x),B(x))} (transport^{A}(p, pr_1(x)), transport^{B}(p, pr_2(x)))$$

Demostración: Por inducción de caminos asumimos $p \equiv refl_x$ y tenemos

$$transport^{A\times B}(p, x) = x$$

$$transport^{A}(p, pr_1(x)) = pr_1(x)$$

$$transport^{B}(p, pr_2(x)) = pr_2(x)$$

Y usando $pair^{=}(x) = (pr_1(x), pr_2(x))$ tenemos el resutado que queremos. □

Finalmente consideremos la funtorialidad de ap en el producto cartesiano. Supongamos que tenemos tipos $A, A', B, B' : \mathcal{U}$. Y funciones $g : A \longrightarrow B$ y $h : A' \longrightarrow B'$. Entonces podemos definir $f : A\times B \longrightarrow A'\times B'$ como $f(x) :\equiv (g(ap_{pr_1}x), h(ap_{pr_2}x))$.

Teorema 3.5.3. *Usando las hipótesis de arriba, y sean* $x, y : A \times B$, $p : pr_1 x = pr_1 y$, $q : pr_2 x = pr_2 y$, *tenemos*

$$f(pair^{=}(p, q)) =_{f(x)=f(y)} pair^{=}(g(p), h(q))$$

Demostración: Primero veamos que las sintaxis está bien tipada. Para esto veamos primero que $pair^{=}(p, q) : x = y$. Es decir, tenemos $f(pair^{=}(p, q)) : f(x) = f(y)$. Por otro lado, dado que $pr_1(f(x)) \equiv g(pr_1 x)$ y $pr_2(f(x)) \equiv h(pr_2 x)$, entonces $pair^{=}(g(p), h(p)) : f(x) = f(y)$.

Ahora por inducción sobre x e y tenemos $x \equiv (a, b)$ e $y \equiv (a', b')$. Luego por inducción sobre $p : a = a'$, $q : b = b'$ tenemos

$$f(pair^{=}(refl_x, refl_x)) =_{f(x)=f(y)} pair^{=}(g(refl_x), h(refl_x))$$

que es trivialmente correcta y habitada por $refl_{(f(x)=f(y))}$, concluyendo la demostración. □

3.5.2. Funciones dependientes

Antes de pasar a formalizar el aspecto homotópico de los tipos de funciones dependientes vale la pena hacer notar dos cosas.

La primera se refiere al hecho de que dado que los $\sum$-tipos son una generalización de los productos cartesianos, entonces de forma similar vamos a tener un par ordenados de caminos, dígase si $x, y : \sum_{z:A} P(z)$ y $p : x = y$ entonces $pr_1(p)$ es un camino similar a los de las secciones anterior. El problema es que $pr_2(p)$ es una trayectoria que "se mueve" sobre varios tipos.

La segunda cosa a notar es que si tenemos $x : A$ y $u, v : P(x)$, de manera que se cumple $(x, u) = (x, v)$ pero no $u = v$, entonces lo único que podemos decir es que se tiene $p : x = x$, tal que se levanta a $p_*(u) = v$. Esto manifiesta el carácter homotópico de la teoría ya que tiene el sentido de que un camino se levanta a algo que termina en un elemento equivalente (homotópico) el espacio total, pero no significa que yacen en la misma fibra. El siguiente teorema enuncia este resultado

Teorema 3.5.4. *Supongamos que* $P : A \longrightarrow \mathcal{U}$ *es una familia sobre* A *y sean* $w, w' : \sum_{x:A} P(x)$. *Entonces tenemos una equivalencia*

$$(w = w') \simeq \sum_{(p:pr_1(w) \equiv pr_1(w'))} p_*(pr_2(w)) = pr_2(w').$$

Demostración: Definimos para cualesquiera $w, w' : \sum_{x:A} P(x)$, una función

$$f : (w = w') \longrightarrow \sum_{(p:pr_1(w) \equiv pr_1(w'))} p_*(pr_2(w)) = pr_2(w')$$

por inducción sobre p tenemos

$$f(w, w, refl_w) :\equiv (refl_{pr_1(w)}, refl_{pr_2(w)}).$$

queremos ver entonces que f es una equivalencia.

Para demostrar el regreso tenemos

$$g : \prod_{(w,w':\sum_{x:A} P(x))} \left(\sum_{(p:pr_1(w) \equiv pr_1(w'))} p_*(pr_2(w)) = pr_2(w') \right) \longrightarrow (w = w')$$

haciendo inducción sobre w y w', resultando en $w \equiv (w_1, w_2)$ y $w' \equiv (w_1', w_2')$, tenemos

$$\left(\sum_{p:w_1 = w_1'} p_*(w_2) = w_2' \right) \longrightarrow ((w_1, w_2) = (w_1', w_2')).$$

Es decir tenemos pares (p, q) donde $p : w_1 = w_1'$ y $q : p_*(w_2) = w_2'$. Haciendo inducción sobre p, tenemos $q : refl_*(w_2) = w_2'$. Así que si demostramos que $(w_1, w_2) = (w_1', w_2')$, terminamos. Tomando inducción sobre q, resulta trivial esto, ya que se reduce a $(w_1, w_2) = (w_1, w_2)$, que esta habitado por $refl_{(w_1,w_2)}$.

De regreso necesitamos ver que $f(g(r)) = r$, para todos w, w' y r, donde r es

$$r : \sum_{p:pr_1(w) = pr_2(w)} (p_*(pr_2(w)) = pr_2(w'))$$

De nuevo diviendo en coordenadas a los caminos w, w' y r, luego hacer doble inducción para reducir las trayectorias coordenadas de r a $refl_{w_i}$, tenemos

$$f(g(refl_{w_1}, refl_{w_2})) = (refl_{w_1}, refl_{w_2})$$

que es correcto por definición.

De igual manera podemos mostrar $g(f(p)) = p$, para todo w, w' y $p : w = w'$. Haciendo inducción sobre p y luego diviendo w en coordenadas tenemos

$$f(g(refl_{w_1}, refl_{w_2})) = (refl_{w_1}, refl_{w_2})$$

que por definición es correcto. Entonces f tiene cuasi-inversa, asi que es equivalencia. □

Deducimos un principio de unicidad, como hicimos con los productos cartesianos, a través del siguiente corolario

Corolario 3.5.1. *Para todo* $z : \sum_{x:A} P(x)$*, podemos escribir* $z = (pr_1(z), pr_2(z))$.

Demostración: Tenemos $refl_{pr_1(z)} : pr_1(z) = pr_1((pr_1(z), pr_2(z)))$, pero usando el teorema anterior, solo necesitamos encontrar un testigo de

$$(refl_{pr_1(z)})_*(pr_2(z)) = pr_2((pr_1(z), pr_2(z)))$$

pero esto es trivial ya que $(refl_{pr_1(z)})_*(pr_2(z)) \equiv pr_2(z) \equiv pr_2((pr_1(z), pr_2(z)))$. □

Hasta ahora sabemos como separar y levantar caminos. Sin embargo necesitamos una operador (equivalente a $pair^{=}$) que nos permita "ensamblarlos". Lo cual va a ser conveniente para establecer propiedades de unicidad y equivalencias como hicimos en epígrafe anterior. Notemos para esto que si $p : x = y$ es un camino, levantado a $u : P(x)$ será $lift(u, p)$, entonces podemos escribir

$$pair^{=}(p, refl_{p_*(u)}) : (x, u) = (y, p_*(u))$$

que es un caso reducido pues estamos tomando $refl_{p_*(u)}$. Sin embargo puede ser extendido si se escribe como algún principio de inducción, esto es

Teorema 3.5.5. *Supongamos que tenemos las familias*

$$P : A \longrightarrow \mathcal{U}$$

y

$$Q : \left(\sum_{x:A} P(x)\right) \longrightarrow \mathcal{U} \ .$$

Entonces podemos construir una familia haciendo la correspondencia

$$x \longmapsto \sum_{u:P(x)} Q(x,u)$$

que está definida sobre A.

Ahora, para cada camino $p : x = y$ *y cada* $(u,z) : \sum_{u:P(x)} Q(x,u)$*, tenemos*

$$p_*(u,z) = \big(p_*(u), pair^{=}(p, refl_{p_*(u)})_*(z)\big)$$

Demostración: Tomemos inducción sobre p. Queremos encontrar un testigo en

$$(refl_x)_*(u,z) = \big((refl_x)_*(u), pair^{=}((refl_x), refl_{(refl_x)_*(u)})_*(z)\big) \tag{3.3}$$

Notemos primero que lo siguiente ocurre

$$(refl_x)_*(u) = u$$

$$pair^{=}((refl_x), refl_u) = refl_{(x,u)}$$

$$(refl_{(x,u)})_*(z) = z$$

por tanto (2.3) se reduce a

$$(refl_x)_*(u,z) = (u,z)$$

que claramente tiene como testigo $refl_{(u,z)}$. □

3.5.3. El tipo unitario

El tipo unitario aunque muy útil para ejemplos y algunas construcciones triviales, es bastante elemental por su estructura simple. La extensión de nuestras propiedades son bastante elementales también.

Teorema 3.5.6. *Para cualesquiera* $x, y : \mathbf{1}$*, se cumple* $x = y \simeq \mathbf{1}$.

Demostración: Sabemos que $* : \mathbf{1}$, entonces es posible definir una función

$$(x = y) \longrightarrow \mathbf{1}$$

enviando todo a $*$.

Por otro lado, para cualesquiera $x, y : \mathbf{1}$, podemos reducir por inducción al caso $x \equiv * \equiv y$ que nos da $refl_* : x = y$. Entonces podemos definir una función constante $\mathbf{1} \longrightarrow (x = y)$.

Es mas o menos claro que ambas son inversas. Tomemos $u : \mathbf{1}$. Por inducción podemos asumir que $u \equiv *$, pero si aplicamos la composición $\mathbf{1} \longrightarrow (x = x) \longrightarrow \mathbf{1}$, nos lleva al mismo resultado.

Y si suponemos $x, y : \mathbf{1}$ y $p : x = y$, tenemos haciendo inducción sobre $p : refl_x$, que la composición $(x = x) \longrightarrow \mathbf{1} \longrightarrow (x = y)$, tambien deriva en $refl_x$. De manera que ambas son cuasi-inversas y por tanto equivalencias. □

En particular cualesquiera dos elementos en $\mathbf{1}$ son iguales. Es decir, el tipo unitario puede escribirse como una familia constante, donde el resultado es un punto único, esto es, sea $A : \mathcal{U}$

$$\mathbf{1} :\equiv \prod_{x:A} *$$

descrito de esta forma el transporte para el tipo unitario es equivalente al de las familias constantes. Esto es

Lema 3.5.1. *Sea* $P : A \longrightarrow \mathcal{U}$ *definida como* $P(x) :\equiv *$ *con* $* : \mathbf{1}$*. Entonces para cualesquiera* $x, y : A$ *y* $p : x = y$ *tenemos un camino*

$$transportconst^*_p(*) : transport^P(p, *) = *.$$

La demostración es trivial.

3.6. $\prod$-tipos y el axioma de extensión funcional

Dado un tipo $A : \mathcal{U}$ y una familia $B : A \longrightarrow \mathcal{U}$ consideramos el tipo de las funciones dependientes $\prod_{x:A} B(x)$. De manera similar a los $\sum$-tipos, esperaríamos que el tipo $f = g$ de caminos de f a g en la familia fuera igual al tipo definido por la igualdad punto a punto, esto es

$$(f = g) \simeq \left(\prod_{x:A} (f(x) =_{B(x)} g(x))\right).$$

Desde el punto de vista tradicional esto queire decir la igualdad tradicional de funciones (son iguales si coinciden en todos los valores de su dominio). Desde el punto de vista topológico esto se refiere a si existe una homotopía entre dichas funciones, es decir un camino continue en el espacio de funciones donde ellas habitan. Y desde el punto de vista categórico quiere decir que un isomorfismo en una categoría funtorial, es una familia de isomorfismos naturales.

El problema que emerge en la teoría de tipos (y por lo cual necesitamos un axioma), es que dada la naturaleza constructiva de esta, es necesario construir un testigo de dicha propiedad. Si recordamos el procedimiento utilizado con los $\sum$-tipos. Notamos que usamos el hecho de que la primera coordenada era siempre un elemento en un tipo que permanecía constante. Esto nos permitia levantar caminos. El problema que emerge aquí es que tal procedimiento no funciona pues no hay estructura invariante en la familia.

Por tanto se establece el siguiente axioma

Axioma 3.6.1 (Extensión funcional)**.** *Para cualesquiera A,B,f y g lo siguiente es una equivalencia*

$$happly : (f = g) \longrightarrow \prod_{x:A} (f(x) =_{B(x)} g(x))$$

En particular el axioma implica que $happly$ tiene una cuasi-inversa

$$funext : \prod_{x:A} (f(x) =_{B(x)} g(x)) \longrightarrow (f = g)$$

que le da nombre al axioma. El sentido de $funext$ es que si una función tiene igualdad punto a punto, entonces esta igualdad puntual puede ser "extendida" a un camino de orden superior (una homotopía). El operador $funext$ es la regla de introducción de el tipo $\prod_{x:A} (f(x) =_{B(x)} g(x)) \longrightarrow (f = g)$, y $happly$ su regla de eliminación. Mientras, se cumple

$$happly(funext(h), x) = h(x), \quad \text{para } h : \prod_{x:A} (f(x) = g(x))$$

y un principio de unicidad proposicional

$$p = funext(x \longmapsto happly(p, x)), \quad \text{para } p : (f = g)$$

Con estas reglas de computación, podemos establecer las propiedades de reflexividad, inversos y composición

- $refl_f = funext(x \longmapsto refl_f(x))$, que se deriva de $happly(refl_f, x) = refl_{f(x)}$.
- $\alpha^{-1} = funext(x \longmapsto happly(\alpha, x)^{-1})$, se deriva de tomar inducción sobre α y $\alpha \equiv refl_f$.
- $\alpha * \beta = funext(x \longmapsto happly(\alpha, x) * happly(\beta, x))$, se deriva de la doble inducción sobre α y β.

Como las funciones no dependientes de tipo $A \longrightarrow B$ son un caso especial de las dependientes $\prod_{x:A} B(x)$, con B variando sobre x, todo lo establecido antes funciona para el caso no dependiente. Las reglas de transporte son un poco mas simples como es de esperarse. Es decir, dado $X : \mathcal{U}$, $p : x_1 =_X x_2$, familias $A, B : X \longrightarrow \mathcal{U}$ y una función $f : A(x_1) \longrightarrow B(x_2)$, tenemos

$$transport^{A \longmapsto B}(p, f) = \left(x \longmapsto transport^B\big(p, f(transport^A(p^{-1}, x))\big) \right) \tag{3.4}$$

donde $x : A(x_2)$ y $A \longmapsto B$ denota abusivamente a la familia $(A \longmapsto B)(x) :\equiv A(x) \longmapsto B(x)$.

Ahroa, similarmente a como hicimos en la sección de productos dependientes, para una familia $p : X \longrightarrow \mathcal{U}$ podemos definir trayectorias dependientes en $p : x =_X y$, desde $u : P(x)$ hasta $v : P(y)$, como $p_*(u) =_{P(y)} v$. Sin embargo cuando P es una familia de tipos de funciones hay una manera mas conveniente de escribir lo anterior.

Lema 3.6.1. *Dadas familias $A, B : X \longrightarrow \mathcal{U}$, un camino $p : x =_X y$ y funciones $f : A(x) \longrightarrow B(x)$ y $g : A(y) \longrightarrow B(y)$, tenemos una equivalencia*

$$(p_*(f) = g) \simeq \prod_{a:A(x)} (p_*(f(a)) = g(p_*(a))).$$

Demostración: Usando inducción asumimos que $p \equiv refl_x$, tenemos entonces

$$((refl_x)_*(f) = g) \simeq \prod_{a:A(x)} ((refl_x)_*(f(a)) = g((refl_x)_*(a)))$$

que se deriva en

$$(f = g) \simeq \prod_{a:A(x)} (f(a) = g(a))$$

que es nuestro axioma. □

3.6.1. Axioma de Univalencia

De la misma forma que establecemos la equivalencia entre dos elementos de un tipo, es posible establecerla sobre los tipos en sí (pues los tipos son elementos de otro tipo). De decir, nos gustaría dada una identidad de la forma $A =_{\mathcal{U}} B$, poder establecer $A \simeq B$. Tenemos una manera elemental de hacer esto

Lema 3.6.2. *Dados tipos $A, B : \mathcal{U}$, hay una función*

$$idtoeqv : (A =_{\mathcal{U}} B) \longrightarrow (A \simeq B).$$

Demostración: La construcción se basa en la aplicación $id_{\mathcal{U}} : \mathcal{U} \longrightarrow \mathcal{U}$. Que se puede pensar como una familia que varía sobre $\mathcal{U}$, y que le asigna a cada tipo $X : \mathcal{U}$ el tipo X mismo. Entonces dado un camino en $\mathcal{U}$, $p : A =_{\mathcal{U}} B$, tenemos una función transportadora $p_* : A \longrightarrow B$. Nos gustaría entonces que p_* fuera una equivalencia. Usando inducción es suficiente asumir $p \equiv refl_A$, pues en tal caso $p_* \equiv id_A$ que es una equivalencia por 3.4.4. Es decir, podemos definir $idtoeqv :\equiv p_*$. □

Sería bueno que el operador $idtoeqv$ fuera una equivalencia pero esto no sucede. Es entonces necesario el siguiente axioma.

Axioma 3.6.2 (Univalencia)**.** *Para cada $A, B : \mathcal{U}$, la función $(A =_{\mathcal{U}} B) \longrightarrow (A \simeq B)$ definida arriba es una equivalencia. Es decir, se tiene*

$$(A =_{\mathcal{U}} B) \simeq (A \simeq B).$$

Digamos que la univalencia se asume para un universo en especial. No tiene sentido hablar de univalencia para el universo de universos. Así que simplemente cuando el axioma se asuma en un universo, se dice que este es univalente. En pocas palabras univalencia significa que tipos iguales pueden ser identificados a traves de una equivalencia. Como hicimos en casos anteriores esto se puede expresar a través de las siguientes reglas definitorias

- Como regla de introducción para $A =_{\mathcal{U}} B$ tenemos,

  $$ua : (A \simeq B) \longrightarrow (A =_{\mathcal{U}} B)$$

  donde ua es una notación que se toma de “univalence axiom“.

- Una regla de eliminación que es $idtoeqv$,

$$idtoeqv \equiv transport^{X \mapsto X} : (A =_{\mathcal{U}} B) \longrightarrow (A \simeq B).$$

- Una regla de computación,

$$transport^{X \mapsto X}(ua(f), x) = f(x).$$

- El principio de unicidad para cada $p : A = B$,

$$p = ua(transport^{X \mapsto X}(p)).$$

También tenemos las propiedades usuales de reflexividad, inversión y concatenación

- $refl_A = ua(id_A)$, que se da por el axioma.
- $ua(f) * ua(g) = ua(f \circ g)$, basta hacer doble inducción sobre f y g.
- $ua(f)^{-1} = ua(f^{-1})$, se deriva de la inducción sobre f y la definición de ua.

Tenemos además el siguiente lema, que no dice como aplicar el axioma de univalencia de manera bastante útil

Lema 3.6.3. *Para cualquier familia $B : A \longrightarrow \mathcal{U}$, $p : x =_A y$ y $u : B(x)$, tenemos*

$$\begin{aligned} transport^{B}(p, u) &= transport^{X \mapsto X}(ap_B(p), u) \\ &= idtoeqv(ap_B(p))(u) \end{aligned}$$

Demostración: La demostración sale del hecho de que

$$transport^{f \circ g}(p, u) \equiv transport^{f}(ap_g(p), u)$$

y la definición de $transport^{X \mapsto X}$. □

3.6.2. Tipo identidad

Supongamos que tenemos elementos $p, q : x =_A y$, entonces la equivalencia de p y q esta dada en el tipo $p =_{x =_A y} q$. Es decir, queremos en esta sección establecer

equivalencias caminos sobre caminos. Esto puede parecer engorroso ya que estamos definiendo estructuras de equivalencia usando la existencia de ellas mismas (cosa que no sucede en la homotopía usual). Veamos como se comporta el mecanismo que hemos desarrollados para los otros tipos en las igualdades.

Teorema 3.6.1. *Si* $f : A \longrightarrow B$ *es una equivalencia, entonces para todo* $a, a' : A$

$$ap_f : (a = a') \longrightarrow (f(a) = f(a'))$$

tambien lo es.

Demostración: Sea f^{-1} cuasi-inversa de f mediante homotopías

$$\alpha : \prod_{b:B} (f(f^{-1}(b)) = b)$$

y

$$\beta : \prod_{b:B} (f^{-1}(f(a)) = a)$$

La cuasi-inversa de ap_f es en esencia

$$ap_f^{-1} : (f(a) = f(a')) \longrightarrow (f^{-1}(f(a)) = f^{-1}(f(a')))$$

Pero aun así debemos hacer uso de la homotopía para poder obtener un elemento $a =_A a'$ de $(f^{-1}(f(a)) = f^{-1}(f(a')))$. Es decir, necesitamos componer con los caminos β_a y β'_a. Es decir si $p : a = a'$, tenemos

$$\beta_a^{-1} * ap_{f^{-1}}(ap_f(p)) * \beta_{a'} = p$$

esto se cumple por la funtorialidad de ap y la naturalidad de las homotopías.

Y de la misma forma vemos que

$$\alpha_f(a) * ap_f(ap_f^{-1}(q)) * \alpha_{f^{-1}(a')} = q$$

lo que nos da la otra relación. □

Consideremos la aplicación de transporte en la familias de trayectorias, es decir $P : A \longrightarrow \mathcal{U}$ donde $P(x)$ es un tipo de identidad. Para esto supongamos primero que $P(x)$ es un tipo de identidad en A con uno de sus extremos (del camino) fijos.

Lema 3.6.4. *Para cada A y $a : A$, con $p : x_1 = x_2$, tenemos*

$$\begin{aligned} transport^{x\longmapsto(a=x)}(p,q) &= q * p && paraq : a = x_1, \\ transport^{x\longmapsto(x=a)}(p,q) &= p^1 * q && paraq : x_1 = a, \\ transport^{x\longmapsto(x=x)}(p,q) &= p^1 * q * p && paraq : x_1 = x_1 \end{aligned}$$

Demostración: Tomando inducción sobre p, lo que nos da $x_1 = x_2$, tenemos

$$transport^{x\longmapsto(x=x)}(refl_x, q) = q * refl_x \equiv q = q$$

$$transport^{x\longmapsto(x=x)}(refl_x, q) = refl_x^1 * q \equiv q = refl_x * q \equiv q$$

$$transport^{x\longmapsto(x=x)}(refl_x, q) = refl_x^1 * q * refl_x \equiv q = refl_x * q * refl_x \equiv q$$

□

Podemos obetener versiones mas generales del lema. Donde los extremos de nuestros caminos no sean constantes y donde las funciones que definen los extremos sean dependientes. Esto se expresa en los siguientes teoremas

Teorema 3.6.2. *Sean $f, g : A \longrightarrow B$, con $p : a =_A a'$ y $q : f(a) =_B g(a)$, tenemos*

$$transport^{x\longmapsto(f(x)=_B g(x))}(p,q) =_{f(a')=g(a')} (ap_f p)^{-1} * q * (ap_g p).$$

Teorema 3.6.3. *Sean $B : A \longrightarrow \mathcal{U}$, $f, g : \prod_{x:A} B(x)$ con $p : a =_A a'$ y $q : f(a) =_B (a)g(a)$, tenemos*

$$transport^{x\longmapsto(f(x)=_{B(x)} g(x))}(p,q) =_{f(a')=g(a')} (apd_f(p))^{-1} * q * (ap_g(p))$$

Demostración: Haciendo inducción sobre p, tenemos

$$transport^{x\longmapsto(f(x)=_{B(x)} g(x))}(refl_x, q) =_{f(a')=g(a')} (apd_f(refl_x))^{-1} * q * (ap_g(refl_x))$$

que es

$$transport^{x\longmapsto(f(x)=_{B(x)} g(x))}(refl_x, q) =_{f(a')=g(a')} (refl_f(x))^{-1} * q * (refl_g(x))$$

que se reduce a $q = q$. □

Terminamos esta parte con otra caracterización de la equivalencia de los caminos dependientes tal vez mas útil

Teorema 3.6.4. *Dados* $p : a =_A a'$, $q : a = a$ *y* $r : a' = a'$, *tenemos*

$$(transport^{x \longmapsto (x=x)}(p, q) = r) \simeq (q * p = p * r)$$

Demostración: La inducción sobre p hace claro que

$$(q = r) \simeq (q * ref_a = refl_a * r)$$

y usando que $q * ref_a = q$ y $refl_a * r = r$, tenemos la prueba. □

3.6.3. Coproductos

La manera de definir las equivalencias en los coproductos es algo distinta a los procedimientos anteriores. El método usual hasta ahora ha sido escribir la relación que se debe cumplir a través de una función, demostrar las cuasi-inversiones y establecer la equivalencia. En los coproductos, como son co-estructuras, la estrategia es distinta ya que los elementos de estos no estan dados como parte de su naturaleza constructiva (declarados en la sintaxis), si no que se definen como resultados de "flechas" que se aplican de elementos estructuralmente independientes al tipo. Estos tipos son llamados tipos **positivos**. En otras palabras, cuando hablamos del tipo $A + B$ no determina nada cual es la naturaleza de A o B, solo que existan aplicaciones $inl(a)$ y $inr(b)$ que sostengan la construcción. Por este hecho utilizaremos un modelo distinto para caracterizar las equivalencias.

Consideremos el coproducto $A + B$ que se representa con las inyecciones $inl : A \longrightarrow A + B$ e $inr : B \longrightarrow A + B$. Como en $A + B$ de alguna forma tenemos dos estructuras distintas (y disjuntas), una vez definida la equivalencia uno esperaría tener las siguientes relaciones

$$(inl(a_1) = inl(a_2)) \simeq (a_1 = a_2) \tag{3.5}$$

$$(inr(b_1) = inr(b_2)) \simeq (b_1 = b_2) \tag{3.6}$$

$$(inl(a) = inr(b)) \simeq \mathbf{0} \tag{3.7}$$

Para esto tomaremos una familia adecuada. Dígase, una vez fijado un elemento $a_0 : A$ o b_0, dado el caso y tenemos

$$(x \longmapsto (inl(a_0) = x)) : A + B \longrightarrow \mathcal{U} \tag{3.8}$$

$$(x \longmapsto (inr(b_0) = x)) : A + B \longrightarrow \mathcal{U} \tag{3.9}$$

Entonces (2.7) y (2.8) caracterizan (2.4)-(2.6).

Tratemos de ver como podemos caracterizar (2.7) (el razonamiento para (2.8) es idéntico). Para esto definimos una familia

$$code : A + B \longrightarrow \mathcal{U}$$

y escribimos el tipo

$$\prod_{x:A+B} ((inl(a_0) = x) \simeq code(x))$$

es decir $code$ tiene que ser capaz de darnos una relación entre a_0 y x, dígase por ejemplo $code(x) \equiv a_0 = x$. Además debe cumplir otras cosas, si queremos satisfacer las ecuaciones (2.4)-(2.6). Es decir

- $code(inl(a)) = (a_0 = a)$
- $code(inr(b)) = \mathbf{0}$

Entonces, a diferencia de los ejemplos anteriores definimos $code$ usando recursión. Esto es

$$code(inl(a)) :\equiv (a_0 = a),$$
$$code(inr(b)) :\equiv \mathbf{0}$$

Así tenemos el siguiente teorema

Teorema 3.6.5. *Para todo* $x : A + B$*, tenemos* $(inl(a_0) = c) \simeq code(x)$.

Demostración:
Primero definamos una función

$$encode : \prod_{x:A} \prod_{p:inl(a_0)=x} code(x)$$

como en la función de arriba p está basada en $inl(a_0)$, uno puede "transportar" $refl_{a_0}$ sobre p, es decir reescribir lo de arriba usando el transporte (teniendo en cuenta que $refl_{a_0} : code(inl(a_0))$.

$$encode(x, p) :\equiv transport^{code}(p, refl_{a_0})$$

Por otro lado definimos

$$decode : \prod_{x:A} \prod_{c:code(x)} (inl(a_0) = x)$$

para caracterizar $decode$ debemos usar el principio de eliminacion del coproducto para saber si $x \equiv inl(a)$ o $x \equiv inr(b)$. Es decir, en el primer caso tenemos

$$x \equiv inl(a),\ code(inl(a)) \equiv (a_0 = a)$$

entonces $c : a_0 = a$, así que podemos definir $decode(inl(a), c) :\equiv app_inl(c) : (inl(a_0) = inl(a))$, y en el segundo caso

$$x \equiv inr(b),\ code(inr(b)) \equiv mathbf0$$

asi que $c : mathbf0$, entonces podemos definir igual $decode(inr(b), c)$. Que por la regla de eliminación del $mathbf0$ nos da un valor bien tipado.

Es fácil ver ahora que $encode$ y $decode$ son cuasi-inversas para todo $x : A + B$. Para la demostración aplicamos inducción sobre $p \equiv refl_{inl(a_0)}$ y $x \equiv inl(a_0)$.

$$\begin{aligned}
decode(x, encode(x, p)) &\equiv decode(inl(a_0), encode(inl(a_0), refl_{inl(a_0)})) \\
&\equiv decode(inl(a_0), transport^{code}(refl_{inl(a_0)}, refl_{a_0})) \\
&\equiv decode(inl(a_0), refl_{a_0}) \\
&\equiv inl(refl_{a_0}) \\
&\equiv refl_{inl(a_0)} \\
&\equiv p
\end{aligned}$$

Por otro lado sea $x : A + B$ y $c : code(x)$. Hay que dividir en casos, si $x \equiv inl(a)$, entonces $c : a_0 = a$ y $decode(x, c) \equiv ap_{inl}(c)$. Así

$$\begin{aligned} encode(x, decode(x, c)) &\equiv transport^{code}(ap_{inl}(c), refl_{a_0}) \\ &= transport^{a \longmapsto (a_0 = a)}(c, refl_{a_0}) \\ &= refl_{a_0} * c \\ &= c \end{aligned}$$

Finalmente si $x \equiv inr(b)$, entonces $c \equiv \mathbf{0}$, así que podemos concluir lo que queramos. □

3.6.4. Números naturales

Los números naturales tienen una estructura similar a los coproductos en el sentido de que el tipo se define por recursión en vez de establecer sus propiedades sintácticamente.

Entonces también podemos usar la metodología de **codificación-decodificación** para establecer las equivalencias en ellos.

Sea $code : \mathbb{N} \longrightarrow \mathbb{N} \longrightarrow \mathcal{U}$, definido por doble recursión sobre $\mathbb{N}$ de la siguiente forma

$$\begin{aligned} code(0, 0) &:\equiv 1 \\ code(succ(m), 0) &:\equiv \mathbf{0} \\ code(0, succ(n)) &:\equiv \mathbf{0} \\ code(succ(m), succ(n)) &:\equiv code(m, n). \end{aligned}$$

En otras palabras, solo son no nulos los valores de $code$ en la diagonal. También definimos una función dependiente por recursión $r : \prod_{n:\mathbb{N}} code(n, n)$ por

$$\begin{aligned} r(0) &:\equiv * \\ r(succ(n)) &:\equiv r(n). \end{aligned}$$

Teorema 3.6.6. *Para todos* $m, n : \mathbb{N}$ *tenemos que* $(m, n) \simeq code(m, n)$.

Demostración: Definimos

$$encode : \prod_{m,n:\mathbb{N}} (m = n) \longrightarrow code(m, n)$$

transportando, tenemos $encode(m, n, p) :\equiv transport^{code(m,-)}(p, r(m))$. Esto solamente quiere decir que si $m = n$ existe una trayectoria y si no el valor de $encode$ es nulo. Por otro lado tenemos

$$decode : \prod_{m,n:\mathbb{N}} code(m, n) \longrightarrow (m = n)$$

que podemos definir por doble inducción sobre m y n. Cuando m y n son 0 ambos llegamos a $mathbf1$ con $code$, así que en este caso tomamos $mathbf1 \longrightarrow (0 = 0)$, es decir $refl_0$. Cuando m es y n es 0 o viceversa, $code(m, n) \equiv mathbf0$, así que el eliminador de $mathbf0$ se encarga de hacer el caso bien tipado. Cuando ambos son sucesores pordemos definir que $decode(succ(m), succ(n), c)$ sea la composición

$$code(succ(m), succ(n)) \equiv code(m, n) \overset{decode(m,n)}{\longrightarrow} (m, n) \overset{ap_{succ}}{\longrightarrow} (succ(m), succ(n)).$$

Veamos que definidos de estas formas los operadores $encode$ y $decode$ son cuasi-inversos, para todo m y n.

Por un lado, si comenzamos con $p : m = n$, entonces por inducción sobre p es suficiente mostrar que

$$decode(n, n, encode(n, n, refl_n)) = refl_n$$

pero por definición $encode(n, n, refl_n) \equiv r(n)$, entonces hay que ver que $decode(n, n, r(n)) = refl_n$. Cosa que podemos probar por inducción sobre n. Si $n \equiv 0$, $decode(0, 0, r(0)) = refl_0$ por definición. En caso de un sucesor, solo hay que ver que $ap_{succ}(refl_n) = refl_{succ(n)}$ y tenemos $decode(n, n, r(n)) = refl_n$.

Por el otro lado, sea $c : code(m, n)$, entonces procediendo por doble inducción sobre m y n, si ambos son 0, entonces $decode(0, 0, c) \equiv refl_0$, y $encode(0, 0, refl_0) \equiv r(0) \equiv *$. Por lo tanto como cada testigo de $mathbf1$ es equivalente a $mathbf*$, tenemos equivalencia a la identidad en $mathbf1$. Si alguno de los dos m o n es 0, entonces tenemos que la recursión nos da 0 por tanto el principio de eliminación de

0 lo anula. Ahora si los dos son sucesores

$$encode(succ(m), succ(n), decode(succ(m), succ(n), c))$$

$$\begin{aligned}
&= encode(succ(m), succ(n), ap_{succ}(decode(m, n, c)) \\
&= transport^{code(succ(m),-)}(ap_{succ}(decode(m, n, c)), r(succ(m))) \\
&= transport^{code(succ(m),succ(-))}(decode(m, n, c), r(succ(m))) \\
&= transport^{code(m,-)}(decode(m, n, c), r(m)) \\
&= encode(m, n, decode(m, n, c)) \\
&= c
\end{aligned}$$

donde en la última igualdad, aplicamos la hipótesis de inducción. □

En particular podemos ver que para todo $m : \mathbb{N}$

$$encode(succ(m), 0) : (succ(m) = 0) \longrightarrow \mathbf{0}$$

lo que demuestra que 0 no es sucesor de níngun natural.

Capítulo 4

Semigrupos y equivalencia homotópica

Durante todo el texto hemos definido y establecido reglas de introducción, eliminación, reducción, computación, etc, que definen las reglas y naturaleza de la teoría homotópica de tipos. En este capítulo veremos un breve ejemplo de como algunas de estas reglas se materializan en un caso particular de estructura algebráica, dígase los semigrupos[1].

Las pruebas y argumentos exhibidos en este capítulo son en su mayoría aplicables a estructuras algebráicas mas complejas como monoides, grupos, anillos, etc. Es intención tal vez de un trabajo futuro recrear las reglas de formación de estos espacios y construir sus reglas de operación en estructuras homotópicas mas complejas como grupos de homotopía u homología.

Establecido dicho lo anterior, definamos el siguiente tipo

Definición 4.0.1. *Dado un tipo A, definimos $SemigroupStr(A)$, la familia de estructuras de semigrupos a partir de A como*

$$SemigroupStr(A) :\equiv \sum_{(m:A\longrightarrow A\longrightarrow A)} \prod_{(x,y,z:A)} m(x, m(y, z)) = m(m(x, y), z).$$

Un semigrupo es un tipo, con una estructura de estas, esto es

$$Semigroup :\equiv \sum_{A:\mathcal{U}} SemigroupStr(A)$$

[1] Ver apéndice C

Donde obviamente en la definición la fórmula $m(x, m(y, z)) = m(m(x, y), z)$, esta codificiando la asociatividad.

Veamos como podemos trabajar con estas estructuras usando el principio de univalencia.

4.1. Equivalencia de levantamientos

En las estructuras algebraicas usuales, que están basadas en conjuntos, uno prueba que una biyección induce un isomorfismo en la estructura, de manera obvia. La biyección en tales casos solo "cambia el símbolo" del operador que define la estructura algebraica. En teoría de tipos este tipo de razonamiento es equivalente a usar univalencia.

Como ya se mencionó, univalencia es la capacidad de intercambiar un tipo por otro equivalente de manera natural, preservando las reglas de construcción (que en este caso se traducen en propiedades del grupo).

Como $SemigroupStr$ es una familia de semigrupos, nos permite tener una acción sobre los caminos para "transportarlos" de un semigrupo a otro, esto es

$$transport^{SemigroupStr}(ua(e)) : SemigroupStr(A) \longrightarrow SemigroupStr(B)$$

donde $ua(e)$ denota un camino dado por la equivalencia del tipo A con el tipo B. Más aun, $transport^{C}(\alpha)$ siempre tiene como cuasi-inversa $transport^{C}(\alpha^{-1})$, así que es una equivalencia. Por tanto la función de arriba también lo es.

El axioma de univalencia no dice entonces que la función $transport^{SemigroupStr}(ua(e))$ y preserva la estructura, sin embargo no dice como actúa en la transformación de un semigrupo en otro. Así que lo diremos de forma explícita. Sea (m, a) una estructura de semigrupo en A, donde m denota el producto en A (un magma) y a denota una prueba de la asociatividad de m, entonces

$$transport^{SemigroupStr}(ua(e), (m, a))$$

es el semigrupo que tenemos que investigar.

Com $SemigroupStr$ esta definido como una función dependiente, usando el Lema 3.5.5 del Capítulo 2 que define el transporte dependiente. Si tomamos en el lema $p :\equiv ua(e)$, $u :\equiv m$ y $z :\equiv a$, tenemos

$$transport^{SemigroupStr}(ua(e), (m, a)) = (m', a')$$

donde $m' : B \longrightarrow B \longrightarrow B$

$$m'(b_1, b_2) :\equiv transport^{X \longmapsto (X \longmapsto X \longmapsto X)}(ua(e), m)(b_1, b_2)$$

y a', la prueba inducida de que m' es asociativa $a' : Assoc(B, m')$

$$a' :\equiv transport^{(X,m) \longmapsto Assoc(X,m)}(pair^{=}(ua(e), refl^{m'}), a)$$

donde $Assoc(X, m)$ es el tipo que habitan las pruebas de asociatividad, esto es $\prod_{(x,y,z:A)} m(x, m(y, z)) = m(m(x, y), z)$.

Ahora m' nos esta diciendo que transportamos m a través de la equivalencia $ua(e)$, que es una trayectoria del tipo A al B. Se podría pensar como aplicar la misma regla m a elementos del tipo B, es decir, el cambio de símbolo que ocurre en los isomorofismos de estructuras basadas en conjuntos. Veamos a que se reduce la ecuación de m' cuando usamos la fórmula **(2.4)** dos veces, vemos que $m'(b_1, b_2)$ es

$$transport^{X \longmapsto X}\Big(ua(e), m\big(transport^{X \longmapsto X}(ua(e)^{-1}, b_1), transport^{X \longmapsto X}(ua(e)^{-1}, b_2)\big)\Big)$$

Notando que $transport^{X \longmapsto X}$ y ua, son cuasi-inversas, lo de arriba se escribe como

$$e\big(m(e^{-1}(b_1), e^{-1}(b_2))\big) \tag{4.1}$$

lo que dice que dados $b_1, b_2 : B$, m' los regresa a A a través de e^{-1} los opera con m y regresa el resultado a B con e. Un procedimiento similar a un *pushback* categórico[2]. El axioma de extensión funcional nos asegura que m' está caracterizada por su comportamiento en estos valores.

Ahora tratamos de encontrar un camino que nos determine una prueba de la asociatividad de m', esto es $Assoc(B, m') : \prod_{(b_1,b_2,b_3:B)} m(b_1, m(b_2, b_3)) = m(m(b_1, b_2), b_3)$.

[2]Ver apéndice B

Procederemos haciendo Ígebra sobre con la equivalencia e y la prueba de m, esto es

$$\begin{aligned}
m'(m'(b_1,b_2),b_3) &= e(m(e^{-1}(m'(b_1,b_2)),e^{-1}(b_3))), && \text{definición de m'}\\
&= e(m(e^{-1}(e(m(e^{-1}(b_1),e^{-1}(b_2)))),e^{-1}(b_3))) && \text{definición de m'}\\
&= e(m(m(e^{-1}(b_1),e^{-1}(b_2)),e^{-1}(b_3))) && \text{ley inversa de e}\\
&= e(m(e^{-1}(b_1),m(e^{-1}(b_2),e^{-1}(b_3)))) && \text{asociatividad de m}\\
&= e(m(e^{-1}(b_1),e^{-1}(m'(b_2,b_3)))) && \text{ley inversa de e}\\
&= m'(b_1,m'(b_2,b_3)). && \text{definición de m'}
\end{aligned}$$

4.2. Igualdad de semigrupos

Usando la definición de la sección anterior y la equivalencia de semigrupos basada en la trayectoria en la familia de espacios, podemos estudiar, si dados dos semigrupos, estos son iguales. Entonces usando el Teorema 3.5.4 tenemos que el tipo de trayectorias $(A,m,a) =_{Semigroup} (B,m',a')$ es igual al tipo de trayectorias de parejas $p \equiv (p_1,p_2)$ donde

$$p_1 : A =_{\mathcal{U}} B$$

$$p_2 : transport^{SemigroupStr}(p_1,(m,a)) = (m',a')$$

Por univalencia $p_1 = ua(e)$ para alguna equivalencia e, así que lo anterior se puede exresar en lenguaje natural como "si dada una trayectoria del tipo A al tipo B, transportar el producto y su ley de asociatividad por esta coincide con la del semigrupo equivalente". Dicho de ua tercera forma, usando el Teorema 3.5.4, extensión de funcionalidad y todo lo anterior p_2 es una pareja de pruebas, dígase la primera

$$\prod_{y_1,y_2:B} e(m(e^{-1}(y_1),e^{-1}(y_2))) = m'(y_1,y_2)$$

que manifiesta el hecho de que m' es equivalante a hacer el pushback con e y m. Y la otra que muestra que a' es igual a la asociatividad inducida por a, en la prueba de la sección anterior, que usando las leyes unitarias de eliminción de e se puede escribir como

$$\prod_{x_1,x_2:A} e(m(x_1,x_2)) = m'(e(x_1),e(x_2)).$$

La conclusión de todo lo anterior es que usando univalencia podemos hacer una equivalencia (algo esotérica) de un camino entre dos semigrupos con la compatibilidad de sus reglas algebraicas. Es decir, podemos identificar una equivalencia entre semigrupos con un isomorfismo entre las estructuras algebraicas que definen sus reglas de construcción.

Apéndice A

Elementos de Homotopía

A.1. Homotopías

Desde el punto de vista clásico (tpológico) una homotopía es una trayectoria en el espacio de funciones continuas entre dos espacios topológicos. Esto es, sean $f, g : X \longrightarrow Y$, entonces llamamos homotopía a una función continua $H : X \times [0,1] \longrightarrow Y$, tal que $\forall x \in X$

$$H(x,0) = f(x)$$

y

$$H(x,1) = g(x)$$

En particular para todo $x \in X$, $H(x,t) : \{x\} \times [0,1] \longrightarrow Y$ es una trayectoria en Y, $f(x) \sim g(x)$.

A.2. Fibraciones

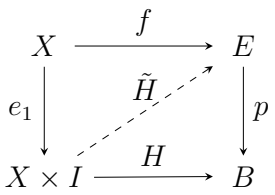
Las fibraciones son aplicaciones que en términos concretos tienen la **propiedad de levantamiento de homotopías**, esto esta definido como

Definición A.2.1. *Sean E, B, X espacios topológicos y sea $p : E \longrightarrow B$ una aplicación continua. Se dice que p tiene la* **propiedad de levantamiento de homotopías** *si dadas $H : X \times I \longrightarrow B$ y $f : X \longrightarrow E$, existe $\tilde{H}$ tal que*

- $p \circ \tilde{H} = H$

- $f = \tilde{H} \circ e$

donde $e : X \longrightarrow X \times I$ *es el encaje trivial. En un diagrama*



Definición A.2.2. *Sean* E, B, X *espacios topológicos. El operador* $p : E \longrightarrow B$ *es una* **fibración** *si tiene la propiedad del levantamiento de homotopías para todo* X*.*

Apéndice B

Elementos de Categorías

B.1. Categorías

Las categorías son un area de las matematicas que intenta estudiar las estructuras abstractas desde un punto de vista mas general. Podemos decir lo siguiente

Definición B.1.1. *Una* **categoría** *C consta de*

- *una clase de* **objetos** $ob(C)$
- *una clase de* **morfismos** $hom(C)$

los morfismos (también llamados **flechas***) pueden verse como un par ordenado que tiene un objeto origen y un objeto destino. Si* a, b *en* $ob(C)$*, podemos escribir* $hom_C(a, b)$ *como los morfismos* $m : a \longrightarrow b$*.*

Sean f, g, h *en* $hom(C)$*, entonces si* $f : a \longrightarrow b$*,* $g : b \longrightarrow c$ *y* $h : c \longrightarrow d$*, se cumple*

- $f \circ (g \circ h) = (f \circ g) \circ h$
- $\forall X$ *en* $ob(C)$*,* $\forall x$ *en* X *se cumple* $\exists id_x$ *en* $hom(C)$*,* $id_x : x \longrightarrow x$ *es un morfismo identidad en el objeto* x*, que es neutro con respecto a la composición, esto es*

  $$id_x \circ f = f$$

  y

  $$f \circ id_x = f$$

  para todo $f \in hom(C)$*.*

B.2. Funtores

Definición B.2.1. *Sean C, D categorías. Un funtor F de C a D. Es un operador que mapea cada objeto X en C a uno W en D, y a cada $f : X \longrightarrow Y$ a $F(f) : F(X) \longrightarrow F(Y)$ cumpliendo las siguientes condiciones*

- $F(id_X) = id_{F(x)}$ *para cada X en C*
- *para cada $f : X \longrightarrow Y$ y $g : Y \longrightarrow Z$ en $hom(C)$, $F(f \circ g) = F(f) \circ F(g)$ en el caso que sea un funtor variante y $F(f \circ g) = F(g) \circ F(f)$ en el caso de uno contravariante.*

Bibliografía

[1] B. Russell. Mathematical logic as based on the theory of types. *American Journal of Mathematics*, 30:222–262, 1908.

[2] A. Church. A set of postulates for the foundation of logic. *Annals of Mathematics, Second Series 34*, 4:839–864, 1933.

[3] P. Martin-Löf. An intuitionistic theory of types: predicative part. *Logic Colloquium '73(Bristol, 1973), Studies in Logic and the Foundations of Mathematics*, 80:73–118, 1975.

[4] The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. http://homotopytypetheory.org/book, Institute for Advanced Study, 2013.

[5] S. Awodey. Type theory and homotopy. *Epistemology versus Ontology: Essays on the Philosophy and Foundations of Mathematics in Honour of Per Martin-Löf*, P. Dybjer et al. (ed.s), Springer:183–201, 2012.

[6] S. Awodey and M. A. Warren. Homotopy theoretic models of identity types. *Mathematical Proceedings of the Cambridge Philosophical Society*, 146(1):45–55, 2009.

[7] S. Awodey. Type theory and homotopy. *arXiv:1010:1810v1*, 2010.

[8] M. A. Warren A. Pelayo, V. Voevodsky. A preliminary univalent formalization of the p-adic numbers. *arXiv:math/1302.1207*, 2013.

[9] M. A. Warren A. Pelayo. Homotopy type theory and voevodsky's univalent foundations. *arXiv:math/1210.5658*, 2012.