En este capítulo daremos una descripción formal y rigurosa de la Teoría de Tipos Dependientes (DTT, en adelante, por sus siglas en inglés), haciendo una comparación a la Matemática Clásica (MC, en adelante) cuando sea conveniente.

0.1. Expresiones, términos y contextos

Dado que las proposiciones y las demostraciones son objetos mismos en DTT, es necesario tener mayor precisión respecto a los significados de ciertos términos.

Definición 0.1.1. Una **signatura** Sg es una colección de símbolos tipográficos. Una **expresión** Exp_{Sg} es una secuencia finita de símbolos provenientes de la signatura Sg.

Ejemplo 0.1.2. La MC utiliza como signatura la colección que contiene a los caracteres alfanuméricos, los conectores lógicos y a los operadores; es decir,

$$Sg = \{\text{`1'}, \text{`('}, \text{`}n\text{'}, \text{`+'}, \text{`}\Sigma\text{'}, \text{`}x_i\text{'}, \dots\}.$$

Una expresión típica de MC es $\langle \text{`1'}, \text{`+'}, \text{`1'}, \text{`='}, \text{`2'} \rangle_{Sg}$.

Notación 0.1.3. Asumiremos siempre que esta misma signatura está implícita en todas las próximas expresiones que escribamos. Además, las expresiones se escribirán sin comillas, sin corchetes y sin hacer referencia a esta signatura.

Justificación. Puesto que la signatura será la misma para el resto de este documento y dado que existe una sola forma de interpretar una expresión simplificada como una secuencia de caracteres, no habrá riesgo de ambigüedad. □

La justificación de esta notación es simple, pero se ha dado para enfatizar que *alguna* justificación es necesaria. En las próximas notaciones, omitiremos las justificaciones triviales y solo justificaremos las más complicadas de realizar.

Para nuestras próximas definiciones, necesitaremos el siguiente concepto.

Definición 0.1.4. Una **metavariable** es una variable que puede tomar como valor cualquier expresión, a excepción de aquellas que contienen alguno de los símbolos '$\equiv$', ':', '$\vdash$' o 'ctx'.

La razón por la que omitimos estos tres símbolos de la definición es porque estos son *caracteres reservados*, los cuales tienen un significado particular y serán introducidos en la próxima sección.

Definición 0.1.5. Sean a y A metavariables, dada una expresión de la forma $((a) : (A))$, diremos que A es un **tipo** y que a es un **término** o **elemento** del tipo A.

Ejemplo 0.1.6. En la expresión $((2 + n) : (\mathbb{N}))$, $\mathbb{N}$ es un tipo y $(2 + n)$ es un término de tipo A.

Notación 0.1.7. En la mayoría de casos, omitiremos los paréntesis, entendiendo que el símbolo ':' tiene menor precedencia que otros símbolos por introducir, a excepción de los símbolos ',' (en contextos), y '$\vdash$' y 'ctx' (en juicios).

Para el resto de este documento utiliaremos metavariables sin mencionar que lo son, dejando la tarea de discernirlas al lector.

Existen ciertas semejanzas entre $a : A$ en DTT y $a \in A$ en MC, pero también hay algunas diferencias importantes. Primero, a no es un elemento que existe independientemente de A; es decir, un término siempre debe estar acompañado del tipo al que corresponde. Relacionado a esto, un término pertenece únicamente a un tipo (con una excepción, ver Sección 0.3), mientras que en teoría de conjuntos un elemento puede pertenecer a varios conjuntos.

Definición 0.1.8. Un **contexto** es una lista finita de expresiones de la forma $a : A$. Es decir, un contexto es una expresión de la forma

$$\langle x_1{:}A_1,\, x_2{:}A_2,\, \ldots,\, x_n{:}A_n\rangle.$$

Los x_i son llamados **variables**.

Ejemplo 0.1.9. La expresión $\langle n{:}\mathbb{N}, v{:}\mathbb{R}^n, M{:}\mathbb{R}^{n\times n}, Mv{:}\mathbb{R}^n\rangle$ es un contexto.

Notación 0.1.10. Omitiremos siempre los corchetes, entendiendo que los símbolos ',' en la lista tienen menor precedencia que otros símbolos por introducir, a excepción de los símbolos '$\vdash$' y 'ctx' en juicios. Además, denotamos el contexto vacío con el símbolo '$\cdot$'.

Nótese que las expresiones y los contextos pueden no estar bien formadas; es decir, pueden ser una secuencia de símbolos sin significado alguno, como $\int 0/0 : \sin\mathbb{Q}$. Los juicios evitan este problema.

0.2. Juicios y reglas de inferencia

Definición 0.2.1. Sea Γ es un contexto, un **juicio** es una expresión de una de las tres siguientes formas:

$$(\Gamma)\ \text{ctx} \qquad (\Gamma) \vdash (a : A) \qquad (\Gamma) \vdash (a \equiv a' : A)$$

Notación 0.2.2. Omitiremos siempre los paréntesis, entendiendo que los símbolos '$\vdash$' y 'ctx' tienen menor precedencia que todos los otros símbolos. En el caso de los últimos dos tipos de juicios, omitiremos la mención del contexto Γ y el símbolo $\vdash$ cuando el contexto no sea relevante o este implícito.

La noción de juicios en DTT toma un rol similar al de proposiciones en MC. El significado de estos juicios es detallado en el siguiente cuadro.

Juicio	Interpretación
Γ ctx	Γ es un contexto bien formado; es decir, una lista de suposiciones bien formadas.
$\Gamma \vdash a : A$	El contexto Γ implica que a es un elemento del tipo A.
$\Gamma \vdash a \equiv a' : A$	El contexto Γ implica que a y a' son objetos iguales por definición del tipo A.

Cuadro 1: Juicios en DTT y su significado.

Notamos que el tercer tipo de juicios se refiere solo a igualdades por definición, en la Sección 0.10 introduciremos otra noción de igualdad. Por otro lado, los primeros dos juicios formalizan la noción de que una expresión tenga "sentido".

Definición 0.2.3. Una expresión de la forma a se dice **bien tipada** si es que existe un contexto Γ y un tipo A tal que Γ ctx y $\Gamma \vdash a : A$.

Similarmente, una expresión de la forma $b : B$ se dice **bien tipada** si es que existe un contexto Γ tal que Γ ctx y $\Gamma \vdash b : B$.

De esta manera, veremos que la expresión previa, $\int 0/0 : \sin \mathbb{Q}$, no está bien tipada. Para llegar a esta conclusión, debemos entender el proceso a través del cual llegamos a estos juicios: la aplicación de reglas de inferencia.

Definición 0.2.4. Una **regla de inferencia** es de la forma

$$\frac{\mathcal{H}_1 \qquad \cdots \qquad \mathcal{H}_k}{\mathcal{C}} \text{ NOMBRE}$$

donde $\mathcal{H}_1, \ldots, \mathcal{H}_k$ y $\mathcal{C}$ son expresiones. Las expresiones $\mathcal{H}_1, \ldots, \mathcal{H}_k$ son llamadas **hipótesis**, mientras que $\mathcal{C}$ es llamada la **conclusión**.

Escribimos a la derecha el nombre de la regla, para ser referenciada posteriormente. Cabe notar que una regla puede tener restricciones adicionales que deben ser corroboradas antes de poder ser aplicada. Si la lista de hipótesis es muy larga, las apilaremos unas sobre otras (ver Reglas 0.7.7). Las reglas toman un rol similar al de la deducción lógica en MC.

Definición 0.2.5. Una **derivación** de un juicio es un árbol invertido con el juicio por derivar en la raíz del árbol, donde el paso de un nodo a otro nodo está justificado por una regla de inferencia.

Ejemplo 0.2.6. Con las reglas que presentaremos posteriormente, el siguiente árbol es una derivación de $\cdot \vdash \mathbf{0} + \mathbf{1} : \mathcal{U}_0$.

$$\dfrac{\dfrac{\dfrac{}{\cdot \text{ ctx}}\text{ ctx-EMP}}{\cdot \vdash \mathbf{0} : \mathcal{U}_0}\text{ 0-FORM} \qquad \dfrac{\dfrac{}{\cdot \text{ ctx}}\text{ ctx-EMP}}{\cdot \vdash \mathbf{1} : \mathcal{U}_0}\text{ 1-FORM}}{\cdot \vdash \mathbf{0} + \mathbf{1} : \mathcal{U}_0}\text{ +-FORM}$$

Con los conceptos previos ya definidos, comenzaremos a introducir las reglas de inferencias de DTT.

0.3. Universos

Como mencionamos previamente, todo término es un elemento de un tipo. Para ser manipulados efectivamente, estos, a su vez, son elementos de otro tipo.

Definición 0.3.1. El tipo de un tipo es llamado un **universo**.

A fin de evitar la paradoja de Russell[1], un universo no es un elemento de sí mismo. Al contrario, existe una infinita jerarquía de universos, la cual se ve formalizada en las siguientes reglas.

Reglas 0.3.2. (Reglas básicas de universos)

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathcal{U}_i : \mathcal{U}_{i+1}}\, \mathcal{U}_i\text{-INTRO} \qquad \frac{\Gamma \vdash A : \mathcal{U}_i}{\Gamma \vdash A : \mathcal{U}_{i+1}}\, \mathcal{U}_i\text{-CUMUL}$$

La primera regla indica que si Γ es un contexto bien formado, entonces Γ implica que el universo $\mathcal{U}_i$ es un elemento del universo $\mathcal{U}_{i+1}$. La segunda regla indica que si Γ implica que A es un elemento del universo $\mathcal{U}_i$, entonces Γ implica que A también es un elemento del universo $\mathcal{U}_{i+1}$.

Dado que son las primeras reglas introducidas, hemos brindado una interpretación de ellas. Para las próximas reglas no realizaremos este tipo de comentarios, salvo para aclarar alguna posible confusión. La lista completa de reglas se encuentra en el Apéndice **??**.

Notación 0.3.3. Omitiremos los subíndices de los universos en la mayoría de escenarios, por lo que interpretaremos expresiones sin sentido como $\mathcal{U} : \mathcal{U}$ agregando índices adecuados, obteniendo $\mathcal{U}_i : \mathcal{U}_{i+1}$. Esta práctica puede traer inconsistencias si no es manejada con precisión, pero la usaremos igualmente para reducir la carga notacional.

Nótese que los subíndices no son elementos del tipo de los naturales, sino son parte del símbolo '$\mathcal{U}_i$'; es decir, consideramos a '$\mathcal{U}_i$' como un solo caracter. Por este motivo, expresiones como $n : \mathbb{N} \vdash A : \mathcal{U}_n$ no están bien formadas.

Con los universos ya definidos, introducimos las reglas respecto a la formación de contextos.

[1] La paradoja de Russell [13] es la siguiente: sea X el conjunto de todos los conjuntos que no se contienen a sí mismos. Si X se contiene a sí mismo, no debería de contenerse a sí mismo por definición. Si X no se contiene a sí mismo, sí debería de hacerlo por definición. Entonces ambos casos llevan a una contradicción.

Reglas 0.3.4. (Reglas básicas de contextos y variables)

$$\frac{}{\cdot \text{ ctx}}\ \text{ctx-EMP} \qquad \frac{x_1{:}A_1, \ldots, x_{n-1}{:}A_{n-1} \vdash A_n : \mathcal{U}_i}{(x_1{:}A_1, \ldots, x_n{:}A_n)\ \text{ctx}}\ \text{ctx-EXT}$$

$$\frac{(x_1{:}A_1, \ldots, x_n{:}A_n)\ \text{ctx}}{x_1{:}A_1, \ldots, x_n{:}A_n \vdash x_i : A_i}\ \text{Vble}$$

donde la regla ctx-EXT tiene la condición adicional de que la variable x_n debe ser distinta a las demás variables $x_1, \ldots, x_{n-1}$, y la regla Vble requiere que $1 \leq i \leq n$.

Nótese que la regla ctx-EMP tiene 0 hipótesis, por lo que siempre es posible aplicarla. A continuación, detallamos el comportamiento de igualdades por definición.

Reglas 0.3.5. (Reglas básicas de igualdades por definición)

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

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash A \equiv B : \mathcal{U}_i}{\Gamma \vdash a : B} \qquad \frac{\Gamma \vdash a \equiv b : A \qquad \Gamma \vdash A \equiv B : \mathcal{U}_i}{\Gamma \vdash a \equiv b : B}$$

Las tres primeras reglas de $\equiv$ indican que esta es una relación de equivalencia, mientras las otras formalizan el buen comportamiento de juicios respecto a tipos iguales por definición.

En las siguientes secciones introduciremos reglas para la formación, introducción y eliminación de algunos constructos. Para cada una de estas reglas, existe una regla correspondiente indicando que estas reglas preservan la igualdad por definición. Como es común es la presentación de reglas de DTT, estas serán omitidas.

0.4. El tipo de funciones

El concepto de una funciones es fundamental en las diferentes áreas de investigación de la Matemática Clásica, en esta sección presentaremos la versión análoga a esta noción en Teoría de Tipos Dependientes.

Definición 0.4.1. Dados dos tipos A y B, el tipo $A \to B$ es llamado el **tipo de funciones** de A a B. Un elemento $f : A \to B$ es llamado una **función**. En este caso, decimos que A es el **dominio** de f y B es el **codominio** de f.

Intuitivamente, para construir una función $f : A \to B$, es suficiente que, dado $x : A$, podamos generar una expresión $b : B$ que esté bien definida, donde b puede contener la variable x dentro de ella. Esto sugiere las siguientes reglas.

Reglas 0.4.2. (Reglas de formación de funciones)

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i}{\Gamma \vdash A \to B : \mathcal{U}_i}\ \to\text{-FORM} \qquad \frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda(x : A).\, b : A \to B}\ \to\text{-INTRO}$$

La expresión $\lambda(x:A).b$ puede entenderse como la función $f : A \to B$ definida por $f(x) = b$, solo que se ha omitido el nombre f. Por este motivo, a veces en la literatura estas funciones se llaman funciones anónimas. En este documento nosotros las llamaremos funciones lambda, el nombre original que les dio Alonzo Church [3].

En la expresión $\lambda(x:A).b$, se dice que x es una **variable ligada** en b. Esto es similar a cómo las expresiones ''$\forall x$' o ''$\int - \mathrm{dx}$' ligan la variable x dentro de estas. Si una variable no es ligada, se dice que es una **variable libre**.

Si f es una función de A en B, entonces podemos aplicarla a un elemento $a : A$ para conseguir un elemento de $b : B$. Intuitivamente, esto se da reemplazando todos las apariciones de x por a. Este proceso se ve formalizado a través de las siguientes reglas.

Reglas 0.4.3. (Reglas de aplicación de funciones)

$$\frac{\Gamma \vdash f : A \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B} \to\text{-ELIM}$$

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda(x:A).b)(a) \equiv b[a/x] : B} \to\text{-COMP}$$

donde la expresión $b[a/x]$ indica que reemplazaremos todas las apariciones libres de x con a.

Nótese que la regla $\to$-COMP implica que la imagen de una función es única, mientras que en MC una función se define como una relación que cumple esta propiedad.

Ejemplo 0.4.4. Para este ejemplo, asumiremos la existencia del tipo de los naturales $\mathbb{N}$, el cual será introducido posteriormente.

Formaremos la función identidad en $\mathbb{N}$ y la aplicaremos en 0. Sea Γ igual a $\mathbb{N}{:}\mathcal{U}$, entonces

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma}{\Gamma, n{:}\mathbb{N}\ \mathrm{ctx}}\ \text{ctx-Ext}}{\Gamma, n{:}\mathbb{N} \vdash n : \mathbb{N}}\ \text{Vble}}{\Gamma \vdash \lambda(n:\mathbb{N}).n : \mathbb{N} \to \mathbb{N}}\ \to\text{-INTRO}$$

Esto muestra que la función existe y tiene el tipo adecuado. Sea Γ igual a $\mathbb{N}{:}\mathcal{U}, 0{:}\mathbb{N}$, veremos que la función se comporta adecuadamente

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma}{\Gamma, n{:}\mathbb{N}\ \mathrm{ctx}}\ \text{ctx-Ext}}{\Gamma, n{:}\mathbb{N} \vdash n : \mathbb{N}}\ \text{Vble} \qquad \dfrac{\Gamma}{\Gamma \vdash 0 : \mathbb{N}}\ \text{Vble}}{\Gamma \vdash (\lambda(n:\mathbb{N}).n)(0) \equiv n[0/n] : \mathbb{N}}\ \to\text{-COMP}}{\Gamma \vdash (\lambda(n:\mathbb{N}).n)(0) \equiv 0 : \mathbb{N}}$$

Definición 0.4.5. Sea A un tipo, la función identidad $\mathsf{id}_A : A \to A$ está definida por

$$\mathsf{id}_A :\equiv \lambda(x:A).x$$

Notación 0.4.6. Usaremos el símbolo $:\equiv$ para dar definiciones. Una definición debe considerarse solo como una abreviación. El símbolo $:\equiv$ no pertenece a DTT per se, sino solo es un mecanismo para reducir la notación en la práctica matemática.

Como mencionamos en la introducción, ya esta función simple representa una mejora respecto a MC. Dado el tipo de todos los conjuntos Set, la función $\mathsf{id}_{\mathsf{Set}}$ es la función identidad en todos los conjuntos, un concepto imposible de formalizar en MC.

Notación 0.4.7. Omitiremos a veces el tipo de una expresión y su contexto asociado, cuando estos no sean relevantes o sean posibles de inferir fácilmente. De esta forma, escribiríamos el juicio derivado en el ejemplo previo como $\mathsf{id}_{\mathbb{N}}(0) \equiv 0$.

Para introducir funciones de varias variables, podríamos introducir el tipo de productos $A \times B$ y definir $f : (A \times B) \to C$. Una alternativa equivalente, pero más conveniente, es introducir el uso de funciones "currificadas" (*curried functions*), nombradas así en honor a Haskell Curry [4].

La currificación de una función $f : (A \times B) \to C$ es una función $f' : A \to (B \to C)$, de tal forma que si $a : A$ y $b : B$, entonces $f'(a)(b) : C$.

Ejemplo 0.4.8. Formaremos una función $A \to (B \to A)$, la cual ignora el valor de la segunda variable, y solo devuelve el valor de la primera variable. Sea $\Gamma :\equiv A{:}\mathcal{U}, B{:}\mathcal{U}$, tenemos

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma}{\Gamma, x{:}A \text{ ctx}}\text{ctx-Ext}}{\Gamma, x{:}A, y{:}B \text{ ctx}}\text{ctx-Ext}}{\Gamma, x{:}A, y{:}B \vdash x{:}A}\text{Vble}}{\Gamma, x{:}A \vdash \lambda(y:B).\, x : B \to A}\to\text{-INTRO}}{\Gamma \vdash \lambda(x:A).\,(\lambda(y:B).\, x) : A \to (B \to A)}\to\text{-INTRO}$$

Poniendo $\Gamma' :\equiv A{:}\mathcal{U}, B{:}\mathcal{U}, a{:}A, b{:}B$, el buen comportamiento es mostrado por

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma'}{\Gamma', x{:}A \text{ ctx}}\text{ctx-Ext}}{\Gamma', x{:}A, y{:}B \text{ ctx}}\text{ctx-Ext}}{\Gamma', x{:}A, y{:}B \vdash a : A}\text{Vble}}{\Gamma', x{:}A \vdash \lambda(y:B).\, a : B \to A}\to\text{-INTRO} \qquad \dfrac{\Gamma'}{\Gamma' \vdash a : A}\text{Vble}}{\dfrac{\Gamma' \vdash \lambda(x:A).\,(\lambda(y:B).\, x)(a) \equiv (\lambda(y:B).\, x)[a/x] : B \to A}{\Gamma' \vdash \lambda(x:A).\,(\lambda(y:B).\, x)(a) \equiv \lambda(y:B).\, a : B \to A}}\to\text{-COMP}$$

y

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma'}{\Gamma', y{:}B \text{ ctx}}\text{ctx-Ext}}{\Gamma', y{:}B \vdash a : A}\text{Vble} \qquad \dfrac{\Gamma'}{\Gamma' \vdash b : B}\text{Vble}}{\dfrac{\Gamma' \vdash (\lambda(y:B).\, a)(b) \equiv a[b/y] : A}{\Gamma' \vdash (\lambda(y:B).\, a)(b) \equiv a : A}}\to\text{-COMP}$$

Juntando estas dos derivaciones, obtenemos que

$$\lambda(x:A).\,(\lambda(y:B).\,x)(a)(b) \equiv (\lambda(y:B).\,a)(b) \equiv a$$

Notación 0.4.9. Cuando puedan ser inferidas o no sean relevantes, omitiremos también el tipo de las variables dentro de una función lambda. Por ejemplo, escribiríamos $\lambda(x:A).\,\lambda(y:B).\,\Phi$ como $\lambda x.\,\lambda y.\,\Phi$.

Si tenemos $f : A_1 \to \cdots \to (A_{n-1} \to A_n)$, escribiremos $f(x_1)(x_2)\cdots(x_n)$ como $f(x_1, x_2, \ldots, x_n)$. Esto no traerá ambigüedad, puesto que el tipo de f indicará si es una función currificada o una función cuyo dominio es un producto de tipos.

Una operación común entre funciones es la composición, la siguiente derivación muestra que esta operación efectivamente existe.

Ejemplo 0.4.10. Sea $\Gamma :\equiv A{:}\mathcal{U}, B{:}\mathcal{U}, C{:}\mathcal{U}$, primero veremos que podemos expandir este contexto adecuadamente; es decir $\Gamma \vdash \Gamma, g{:}B \to C, f{:}A \to B, x{:}A$ ctx.

$$\dfrac{\dfrac{\dfrac{\Gamma}{\Gamma \vdash B : \mathcal{U}_i \text{ ctx}}\,\text{Vble} \qquad \dfrac{\Gamma \text{ ctx}}{\Gamma \vdash C : \mathcal{U}_i \text{ ctx}}\,\text{Vble}}{\Gamma \vdash B \to C{:}\mathcal{U}_i}\,{\to}\text{-FORM}}{\Gamma, (g{:}B \to C) \text{ ctx}}\,\text{ctx-Ext}$$

Además,

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\Gamma, (g{:}B \to C) \text{ ctx}}{\Gamma, (g{:}B \to C) \vdash A : \mathcal{U}_i}\,\text{Vble} \qquad \dfrac{\Gamma, (g{:}B \to C) \text{ ctx}}{\Gamma, (g{:}B \to C) \vdash B : \mathcal{U}_i}\,\text{Vble}}{\Gamma, (g{:}B \to C) \vdash A \to B{:}\mathcal{U}_i}\,{\to}\text{-FORM}}{\Gamma, (g{:}B \to C), (f{:}A \to B) \text{ ctx}}\,\text{ctx-Ext}}{\Gamma, (g{:}B \to C), (f{:}A \to B) \vdash x : A}\,\text{Vble}}{\Gamma, (g{:}B \to C), (f{:}A \to B), (x : A) \text{ ctx}}\,\text{ctx-Ext}$$

Ahora, tenemos que

$$\dfrac{\Gamma, (g{:}B \to C), (f{:}A \to B), (x{:}A) \text{ ctx}}{\Gamma, (g{:}B \to C), (f{:}A \to B), (x{:}A) \vdash f : A \to B}\,\text{Vble}$$

$$\dfrac{\Gamma, (g{:}B \to C), (f{:}A \to B), (x{:}A) \text{ ctx}}{\Gamma, (g{:}B \to C), (f{:}A \to B), (x{:}A) \vdash x : A}\,\text{Vble}$$

Juntando estas dos derivaciones, obtenemos

$$\dfrac{\Gamma, \ldots, (x{:}A) \vdash f : A \to B \qquad \Gamma, \ldots, (x{:}A) \vdash x : A}{\Gamma, (g{:}B \to C), (f{:}A \to B), (x{:}A) \vdash f(x) : B}\,{\to}\text{-ELIM}$$

Similarmente, tenemos que

$$\dfrac{\Gamma, (g{:}B \to C), (f{:}A \to B), (x{:}A) \text{ ctx}}{\Gamma, (g{:}B \to C), (f{:}A \to B), (x{:}A) \vdash g : B \to C}\,\text{Vble}$$

Finalmente, llegamos a

$$\dfrac{\dfrac{\dfrac{\dfrac{\Gamma,\ldots,(x{:}A)\vdash g : B\to C \qquad \Gamma,\ldots,(x{:}A)\vdash f(x) : B}{\Gamma,(g{:}B\to C),(f{:}A\to B),(x{:}A)\vdash g(f(x)) : C}\to\text{-ELIM}}{\Gamma,(g{:}B\to C),(f{:}A\to B)\vdash \lambda(x:A).\, g(f(x)) : A\to C}\to\text{-INTRO}}{\Gamma,(g{:}B\to C)\vdash \lambda f.\,(\lambda(x:A).\, g(f(x))) : (A\to B)\to(A\to C)}\to\text{-INTRO}}{\Gamma\vdash \lambda g.\,\lambda f.\,(\lambda(x:A).\, g(f(x))) : (B\to C)\to(A\to B)\to(A\to C)}\to\text{-INTRO}$$

Esto muestra que la función

$$\begin{aligned}&\mathsf{comp}_{A,B,C} : (B\to C)\to(A\to B)\to(A\to C)\\ &\mathsf{comp}_{A,B,C} :\equiv \lambda(g:B\to C).\,\lambda(f:A\to B).\,(\lambda x{:}A.\, g(f(x)))\end{aligned}$$

existe. La prueba de que esta función se comporta como esperaríamos, es decir que $\mathsf{comp}_{A,B,C}(g,f,x)\equiv g(f(x))$, es semejante a la del Ejemplo 0.4.8, y por lo tanto la omitiremos.

También se puede verificar el hecho conocido de que la composición es asociativa; es decir, que se tiene

$$\mathsf{comp}_{A,C,D}(h,\mathsf{comp}_{A,B,C}(g,f))\equiv \mathsf{comp}_{A,B,D}(\mathsf{comp}_{B,C,D}(h,g),f)$$

Con esta operación de composición de funciones entonces obtenemos una categoría.

Definición 0.4.11. La categoría Type tiene como objetos tipos, y como morfismos funciones entre tipos. El morfismo identidad asociado a un objeto A está dado por la función identidad id_A.

La última regla de funciones indica que si formamos una nueva función lambda, que recibe x y devuelve $f(x)$ entonces esta es la misma función que la f original.

Reglas 0.4.12. (Principio de unicidad para funciones)

$$\dfrac{\Gamma\vdash f : A\to B}{\Gamma\vdash f\equiv(\lambda(x:A).\, f(x)) : A\to B}\to\text{-UNIQ}$$

Nótese que esta regla no dice que si tenemos dos funciones $f,g : A\to B$ tales que $f(x)\equiv g(x)$ para todo x, entonces $f\equiv g$. Esta propiedad, apropiadamente modificada, será introducida en la Sección **??**.

0.5. El tipo de funciones dependientes

En MC, una práctica común es utilizar sucesiones infinitas de elementos x_i de un conjunto X. Formalmente, esto corresponde a una función $f : \mathbb{N}\to X$. Similarmente, a veces es necesario indexar no solo elementos, sino conjuntos con cierta estructura por otros conjuntos.

Por ejemplo, a cada punto p en una variedad M, le corresponde su espacio tangente T_pM. Se entiende que esta última construcción está asociada a cierta función $g : M \to \bigcup_{p\in M}\{T_pM\}$. En este último caso, es común decir que $\{T_pM\}_{p\in M}$ es una familia de conjuntos indexada por $p \in M$. Presentamos el constructo análogo en DTT.

Definición 0.5.1. Si tenemos que $\Gamma, x : A \vdash B : \mathcal{U}$, entonces diremos que B es una **familia de tipos** indexada por $x : A$.

Recordamos que B no es el carácter 'B', sino una metavariable, por lo que esta puede contener la variable x dentro de sí. Además, notamos que el requerimiento que $\Gamma, x : A \vdash B : \mathcal{U}$ es equivalente a que exista un $f : A \to \mathcal{U}$, por lo que en este caso también diremos que B es una familia de tipos.

Ejemplo 0.5.2. Cada punto x en un espacio topológico X tiene asociado un grupo, su grupo fundamental. Es decir, $\Gamma, x : X \vdash \pi_1(x, X) : \mathcal{U}$.

Ejemplo 0.5.3. Sea $B : \mathcal{U}$ un tipo, para todo $a : A$, se tiene $\Gamma, a : A \vdash B : \mathcal{U}$. En este caso, decimos que B es una **familia constante**.

Nótese que en el caso de familias constantes, una función $f : A \to B$ asigna a cada $a : A$ un elemento $f(a) : B$. Esto se puede generalizar para familias arbitrarias.

Ejemplo 0.5.4. Sea X un espacio topológico, para cada $x : X$ podemos escoger un elemento de $\pi_1(x, X) : \mathcal{U}$, el elemento neutro de ese grupo $0_{\pi_1(x,X)}$.

En este último ejemplo tenemos una regla de correspondencia, por lo que se esperaría que podemos formar una función f con el tipo $X \to \pi_1(x, X)$. Sin embargo, este tipo previo no está bien tipado, el x que aparece en el codominio no está definido. El problema es que el codominio depende del $x : X$ escogido en el dominio, para estos casos necesitaremos el tipo de **funciones dependientes** de A en B, denotado por $\prod_{(x:A)} B$.

Reglas 0.5.5. (Reglas de formación de funciones dependientes)

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

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

donde la expresión $\prod_{(x:A)} B$ liga a x hasta el final de esta.

Con estas reglas, se puede definir el tipo $\prod_{(x:X)} \pi_1(x, X)$, y una de las funciones pertenecientes a este tipo sería $\lambda(x : X).\, 0_{\pi_1(x,X)}$.

Igual que en el caso de funciones no dependientes, tenemos reglas que nos indican el proceso de aplicación de estas funciones, así como un principio de unicidad.

Reglas 0.5.6. (Reglas de aplicación de funciones)

$$\frac{\Gamma \vdash f : \prod_{(x:A)} B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B[a/x]}\ \Pi\text{-ELIM}$$

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda(x:A).b)(a) \equiv b[a/x] : B[a/x]}\ \Pi\text{-COMP}$$

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

Notamos que estas reglas son generalizaciones directas de las reglas de funciones introducidas en la sección anterior. En efecto, tomaremos estas como las reglas oficiales y definiremos el tipo $A \to B$ como $\prod_{(x:A)} B$; puesto que en este caso B es una familia constante, y se tiene que $B[a/x]$ es B, con lo que se obtienen las reglas originales.

Notación 0.5.7. Los variables ligadas por Π tienen menor precedencia que otros operadores dentro de un tipo, por lo que $\prod_{(x:A)} B \to C$ se entiende como $\prod_{(x:A)}(B \to C)$, por ejemplo. Además, para acentuar el énfasis de la (posible) dependencia de B sobre x, escribiremos $\prod_{(x:A)} B(x)$, entendiendo por esto como simplemente $\prod_{(x:A)} B$. Finalmente, mencionamos que existe otra notación alternativa para $\prod_{(x:A)} B(x)$, como

$$\prod_{x:A} B(x) \quad \text{y} \quad \Pi(x:A), B(x).$$

El uso del símbolo Π sugiere que este tipo puede ser interpretado también como el producto cartesiano de lo B_i, como lo muestra el próximo ejemplo. La siguiente discusión es informal, y utilizaremos algunos términos de teoría de conjuntos para ayudar la exposición. Sin embargo, estas nociones no están definidas para DTT en este punto.

Ejemplo 0.5.8. Sea A un tipo con tres elementos a_1, a_2, a_3 y sean X, Y, Z tres tipos con m, n y p elementos respectivamente. Podemos asignar al elemento a_1 el tipo X, al elemento a_2 el tipo Y y al elemento a_3 el tipo Z. Esto nos da una familia de tipos $B : A \to \mathcal{U}$.

Una función $f : \prod_{(x:A)} B$ asigna a cada $x : A$ un elemento de $B(x)$.

$x : A$	$B(x)$
a_1	$B(a_1) \equiv X \equiv \{x_1, \ldots, x_m\}$
a_2	$B(a_2) \equiv Y \equiv \{y_1, \ldots, y_n\}$
a_3	$B(a_3) \equiv Z \equiv \{z_1, \ldots, z_p\}$

Definamos la función $f_{i,j,k} : \prod_{(x:A)} B(x)$ por $f(a_1) = x_i, f(b_1) = y_j, f(a_3) = z_k$ donde $1 \leq i \leq m$, $i \leq j \leq n$, $1 \leq k \leq p$. De esta forma, vemos que los elementos de $\prod_{(x:A)} B(x)$ corresponden exactamente a triples (x, y, z) con $x : X, y : Y$ y $z : Z$.

Así, el número de funciones en $\prod_{(x:A)} B(x)$, es el número de elecciones posibles para cada a_i, es decir

$$\left|\prod_{x:A} B(x)\right| = \prod_{x:A} |B(x)| = |X| \cdot |Y| \cdot |Z|$$

El hecho de que $|A \to B| = |B|^{|A|}$ es un caso particular de esto.

Una pequeña limitación de la función $\mathsf{id}_A : A \to A$ es que está definida solo para el tipo A. Formalmente, tendríamos que crear una nueva función id_T para cada tipo T. Las funciones dependientes evitan la repetición de este proceso.

Ejemplo 0.5.9. Existe una función id que asigna a cada $A : \mathcal{U}$, su función identidad $\mathsf{id}_A : A \to A$.

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A : \mathcal{U}\ \text{ctx}}{A : \mathcal{U}, x : A\ \text{ctx}}\ \text{ctx-Ext}}{A : \mathcal{U}, x : A \vdash x : A}\ \text{Vble}}{A : \mathcal{U} \vdash \lambda(x:A).\, x : A \to A}\ \to\text{-INTRO}}{\cdot \vdash \lambda(A:\mathcal{U}).\, \lambda(x:A).\, x : \prod_{(A:\mathcal{U})} A \to A}\ \Pi\text{-INTRO}$$

La derivación de su buen comportamiento es similar al Ejemplo 0.4.8, y será omitida.

Dados estos últimos ejemplos, vemos que dada una función lambda bien tipada, es un proceso simple (pero tedioso) generar la derivación de su existencia y buen comportamiento. Por este motivo, en las secciones siguientes comenzaremos a escribir solamente la función lambda, omitiendo las derivaciones asociadas.

Notación 0.5.10. Es común en MC definir funciones a partir de reglas de correspondencia; es decir, definiendo f por su comportamiento en un x arbitrario. Utilizaremos esta misma práctica, entendiendo estas definiciones como una abreviación de funciones lambda. Por ejemplo, la definición de la función

$$\mathsf{id} : \prod_{A:\mathcal{U}} A \to A$$
$$\mathsf{id} :\equiv \lambda(A:\mathcal{U}).\, \lambda(x:A).\, x$$

Puede ser reescrita simplemente como

$$\mathsf{id} : \prod_{A:\mathcal{U}} A \to A$$
$$\mathsf{id}(A, x) :\equiv x$$

Finalmente, para funciones currificadas de varios parámetros, se asociaran aplicaciones repetidas de '$\to$' por la derecha, por lo que $A \to B \to C$ se entiende como $A \to (B \to C)$, por ejemplo. Todos los demás constructos por introducir asociarán hacia la izquierda, de tal forma que $A \times B \times C$ es $(A \times B) \times C$.

Ejemplo 0.5.11. Existe una función swap que invierte el orden de las variables de una función de dos parámetros.

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

Y esta está definida por

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

Notamos que de acuerdo a la notación anterior, esta también pudo haber sido definido como:

$$\mathsf{swap}(A, B, C, f, b, a) :\equiv f(a, b)$$

Como se esperaría, podemos generalizar la definición de composición de funciones del Ejemplo 0.4.10.

Definición 0.5.12. Podemos definir la función de composición de funciones

$$\mathsf{comp}^* :\equiv \prod_{(A:\mathcal{U})} \prod_{(B:\mathcal{U})} \prod_{(C:\mathcal{U})} \left(\prod_{y:B} C(y) \right) \to \prod_{(f:A\to B)} \prod_{(x:X)} C(f(x))$$

dada por

$$\mathsf{comp}^*(A, B, C, g, f, x) :\equiv g(f(x))$$

Esta función generaliza a la función del Ejemplo 0.4.10, en el sentido de que

$$\mathsf{comp}_{A,B,C}(g, f) \equiv \mathsf{comp}^*(A, B, C, g, f)$$

Como es usual, escribiremos $g \circ f$ en vez de $\mathsf{comp}^*(A, B, C, g, f)$, dejando los tipos A, B y C implícitos.

0.6. El tipo de pares dependientes

Así como el tipo de funciones $A \to B$ es un caso particular del tipo de funciones dependientes $\prod_{(x:A)} B(x)$, el tipo de pares $A \times B$ es un caso particular del tipo de pares dependientes $\sum_{(x:A)} B$ donde B puede depender de $x : A$.

Intuitivamente, dados dos elementos $a : A$ y $b : B(a)$ es posible generar el par (a, b). Las siguientes reglas formalizan esto[2].

Reglas 0.6.1. (Reglas de formación de pares dependientes)

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : A \to \mathcal{U}_i}{\Gamma \vdash \sum_{(x:A)} B : \mathcal{U}_i}\ \Sigma\text{-FORM}$$

$$\frac{\Gamma \vdash B : A \to \mathcal{U}_i \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B(a)}{\Gamma \vdash (a, b) : \sum_{(x:A)} B}\ \Sigma\text{-INTRO}$$

[2]El uso del tipo de funciones en estas reglas podría hacer parecer que estas tienen un rol más fundamental que el de pares dependientes; sin embargo, es posible formalizar estas reglas sin hacer mención a las funciones, ver [17, Apéndice A.2.]

donde la expresión $\sum_{(x:A)} B$ liga a x hasta el final de esta.

Notación 0.6.2. En caso B sea una familia constante, escribiremos también $A \times B$ en vez de $\sum_{(x:A)} B$. Los variables ligadas por Σ tienen menor precedencia que otros operadores dentro de un tipo, por lo que $\sum_{(x:A)} B \to C$ se entiende como $\sum_{(x:A)}(B \to C)$, por ejemplo.

En el caso del producto '$\times$', este tiene mayor precedencia que '$\to$', por lo que $A \times B \to C$ se interpreta como $(A \times B) \to C$. Lo mismo aplica para el coproducto '$+$' definido en la próxima sección.

Además, para acentuar el énfasis de la (posible) dependencia de B sobre x, escribiremos $\sum_{(x:A)} B(x)$, entendiendo por esto como simplemente $\sum_{(x:A)} B$. Finalmente, mencionamos que existe otra notación alternativa para $\sum_{(x:A)} B(x)$, como

$$\sum_{x:A} B(x) \quad \text{y} \quad \Sigma(x:A), B(x).$$

Similarmente al caso de funciones dependientes, el uso del símbolo Σ sugiere que este tipo puede ser interpretado también como una suma (disjunta) de los B_i, como lo muestra el próximo ejemplo informal.

Ejemplo 0.6.3. Sea $A = \{a_1, a_2, a_3\}$ y sean X, Y, Z, B como en el Ejemplo 0.5.8. El tipo de pares dependientes $\sum_{(x:A)} B$ contiene pares (a, b) con $a : A$ y $b : B(a)$. Por ejemplo, podemos definir un par $(a_1, x_i) : \sum_{(x:A)} B$.

De esta forma, los elementos de $\sum_{(x:A)} B$ corresponden exactamente a elementos de la unión disjunta de los elementos de X, Y y Z. En efecto, tenemos

$$\left|\sum_{(x:A)} B\right| = \Sigma_{x:A}|B(x)| = |X| + |Y| + |Z|$$

Las reglas previas solo muestran cómo formar pares, para que estos sean útiles es necesario ver cómo se pueden usar; es decir, cómo formar funciones cuyo dominio sea un tipo de pares dependientes.

Reglas 0.6.4. (Reglas de eliminación de pares dependientes.)

$$\frac{\Gamma \vdash C : (\sum_{(x:A)} B) \to \mathcal{U}_i \qquad \Gamma \vdash g : \prod_{(a:A)} \prod_{(b:B(a))} C((a,b)) \qquad \Gamma \vdash p : \sum_{(x:A)} B}{\Gamma \vdash \mathsf{ind}^{C,g,p}_{\sum_{(x:A)} B} : C(p)} \ \Sigma\text{-ELIM}$$

$$\frac{\Gamma \vdash C : (\sum_{(x:A)} B) \to \mathcal{U}_i \qquad \Gamma \vdash g : \prod_{(a:A)} \prod_{(b:B(a))} C(a,b) \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B[a/x]}{\Gamma \vdash \mathsf{ind}^{C,g,(a,b)}_{\sum_{(x:A)} B} \equiv g(a)(b) : C((a,b))} \ \Sigma\text{-COMP}$$

Aquí, $\mathsf{ind}^{C,g,p}_{\sum_{(x:A)} B}$ es un *primitivo*, un elemento que afirmamos existe, y es un elemento de $C(p)$. En otras palabras, para cada C, g y p que satisfagan los requisitos de la regla, Σ-ELIM indica que exite este primitivo, el cual depende de estas tres metavariables. Similarmente para la regla Σ-COMP.

Estas dos reglas nos dan el siguiente principio de inducción para pares dependientes.

Teorema 0.6.5. *Sea A un tipo, $B : A \to \mathcal{U}$ una familia de tipos sobre A, entonces existe una función*

$$\mathsf{ind}_{\sum_{(x:A)} B(x)} : \prod_{(C:(\sum_{(x:A)} B(x))\to\mathcal{U})} \left(\prod_{(a:A)}\prod_{(b:B(a))} C((a,b))\right) \to \prod_{(p:\sum_{(x:A)} B(x))} C(p)$$

tal que

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

Demostración. Definimos la función por

$$\mathsf{ind}_{\sum_{(x:A)} B(x)}(C, g, p) :\equiv \mathsf{ind}^{C,g,p}_{\sum_{(x:A)} B} : C(p),$$

el cual existe por la regla Σ-ELIM, mientras que la regla Σ-COMP nos indica que se comporta apropiadamente en pares. □

El principio de inducción para pares dependientes captura la adjunción Hom (Ejemplo **??**):

$$\mathrm{Hom}(A \times B, C) \cong \mathrm{Hom}(A, \mathrm{Hom}(B, C))$$

En efecto, es esta equivalencia la que aprovechamos para nuestra definición de funciones currificadas.

Veamos como podemos utilizar este principio para mostrar la existencia de las funciones de proyección en cada uno de las coordenadas.

Teorema 0.6.6. *Sea A un tipo y B una familia de tipos sobre A. Entonces existen funciones pr_1 y pr_2 tales que $\mathsf{pr}_1(a,b) = a$ y $\mathsf{pr}_2(a,b) = b$.*

Demostración. Sean

$$C :\equiv \lambda p.\, A : \sum_{x:A} B(x) \to \mathcal{U}$$
$$g :\equiv \lambda x.\, \lambda y.\, x : \prod_{(a:A)} \prod_{(b:B(a))} A$$

Notando que $C(a,b) \equiv A$, y aplicando el principio de inducción obtenemos

$$\mathsf{pr}_1 :\equiv \mathsf{ind}_{\sum_{(x:A)} B(x)}(C, g) : \prod_{p:\sum_{(x:A)} B(x)} A$$
$$\mathsf{pr}_1(a,b) :\equiv \mathsf{ind}_{\sum_{(x:A)} B(x)}(C, g, (a,b)) \equiv g(a)(b) \equiv a$$

De manera similar, tomando

$$C' :\equiv \lambda p.\, B(\mathsf{pr}_1(p)) : \sum_{x:A} B(x) \to \mathcal{U}$$
$$g :\equiv \lambda x.\, \lambda y.\, y : \prod_{(a:A)} \prod_{(b:B(a))} B(a)$$

y notando que $C(a,b) \equiv B(\mathsf{pr}_1((a,b))) \equiv B(a)$, aplicando el principio de inducción obtenemos

$$\mathsf{pr}_2 :\equiv \mathsf{ind}_{\sum_{(x:A)} B(x)}(C', g') : \prod_{p:\sum_{(x:A)} B(x)} B(\mathsf{pr}_1(p))$$
$$\mathsf{pr}_2 :\equiv \mathsf{ind}_{\sum_{(x:A)} B(x)}(C', g', (a,b)) \equiv g'(a,b) \equiv b$$

Por lo que ambas proyecciones existen. □

Aplicar directamente el principio de inducción requiere cierto nivel de cuidado, en MC clásica definiríamos la función pr_2 simplemente como $\mathsf{pr}_2((a,b)) :\equiv b$. Veremos que esta práctica también se puede realizar en DTT.

Notación 0.6.7. (Búsqueda de patrones para pares dependientes)
Sea A un tipo, $B : A \to \mathcal{U}$ una familia de tipos sobre A y $C : (\sum_{(x:A)} B(x)) \to \mathcal{U}$. Podemos definir una función $f : \prod_{(p:\sum_{(x:A)} B(x))} C(p)$ por

$$f((a,b)) :\equiv \Phi$$

donde $\Phi : C((a,b))$ es una expresión que puede contener a o b.

Justificación. Dados $a : A$, $b : B(a)$ y $\Phi : C((a,b))$, podemos definir

$$g :\equiv \lambda(a:A).\lambda(b:B(a)).\Phi : \prod_{(a:A)} \prod_{(b:B(a))} C((a,b))$$

Entonces,

$$f :\equiv \mathsf{ind}_{\sum_{(x:A)} B(x)}(C, g) : \prod_{p:\sum_{(x:A)} B(x)} C(p)$$

es tal que $f((a,b)) \equiv \Phi$. □

Esto justifica el hecho "obvio" de que basta definir una función en un par arbitrario para que la función esté bien definida en todo el tipo de pares dependientes.

Notación 0.6.8. Omitiremos a veces los paréntesis de un par cuando estos estén dentro de una aplicación de una función o dentro de otro par. Por ejemplo, escribiremos $C(a,b)$ en vez de $C((a,b))$, y (a,b,c) en vez de $(a,(b,c))$ o $((a,b),c)$.

0.7. 0, 1, 2 y el tipo del coproducto

El tipo 0

El tipo $\mathbf{0} : \mathcal{U}$ representa el tipo vacío, por lo que si obtuviésemos un elemento de él, esto sería un absurdo, y podríamos concluir cualquier cosa. Las siguientes reglas capturan esta intuición.

Reglas 0.7.1. (Reglas del tipo $\mathbf{0}$.)

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathbf{0} : \mathcal{U}_0}\ \mathbf{0}\text{-FORM} \qquad \frac{\Gamma \vdash C : \mathbf{0} \to \mathcal{U}_0 \qquad \Gamma \vdash z : \mathbf{0}}{\Gamma \vdash \mathsf{ind}_{\mathbf{0}}^{C,z} : C(z)}\ \mathbf{0}\text{-ELIM}$$

Aquí, al igual que para el caso de los pares dependientes, $\mathsf{ind}_{\mathbf{0}}^{C,z}$ es un primitivo, el cual afirmamos que existe siempre que se satisfagan las condiciones necesarias. Esto mismo aplica para los elementos $\mathsf{ind}_{\mathbf{1}}^{C,c,a}$, inl_{A+B}^{a} y inr_{A+B}^{b} introducidos posteriormente en esta sección.

El principio de inducción del $\mathbf{0}$ implica que, dado un $z : \mathbf{0}$, podemos generar un elemento de un tipo que (posiblemente) depende de z.

Teorema 0.7.2. *Existe una función*

$$\mathsf{ind}_{\mathbf{0}} : \prod_{(C:\mathbf{0}\to\mathcal{U})} \prod_{(z:\mathbf{0})} C(z)$$

Demostración. Definimos la función por

$$\mathsf{ind}(C, z) :\equiv \mathsf{ind}_{\mathbf{0}}^{C,z}.$$

□

Nótese que, en particular, el principio de inducción del tipo $\mathbf{0}$ nos da una función $\mathsf{rec}_{\mathbf{0}}(C) : \mathbf{0} \to C$, para todo tipo C. En efecto, podemos definir

$$\mathsf{rec}_{\mathbf{0}}(C, x) :\equiv \mathsf{ind}_{\mathbf{0}}(\lambda x.\, C, x)$$

Esto nos da indicios de que el tipo $\mathbf{0}$ es un objeto inicial de la categoría Type, y este es efectivamente el caso; pero la demostración de la unicidad de la función $\mathsf{rec}_{\mathbf{0}}(C)$ tendrá que esperar (ver Proposición **??**).

El tipo 1

De manera análoga, el tipo $\mathbf{1} : \mathcal{U}$ representa el tipo con un solo elemento $\star$, por lo que para generar una función con dominio es suficiente definirla en $\star$. Las siguientes reglas dicen precisamente esto.

Reglas 0.7.3. (Reglas del tipo $\mathbf{1}$.)

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i}\ \mathbf{1}\text{-FORM} \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash \star : \mathbf{1}}\ \mathbf{1}\text{-INTRO}$$

$$\frac{\Gamma \vdash C : \mathbf{1} \to \mathcal{U}_i \qquad \Gamma \vdash c : C(\star) \qquad \Gamma \vdash a : \mathbf{1}}{\Gamma \vdash \mathsf{ind}_{\mathbf{1}}^{C,c,a} : C(a)}\ \mathbf{1}\text{-ELIM}$$

$$\frac{\Gamma \vdash C : \mathbf{1} \to \mathcal{U}_i \qquad \Gamma \vdash c : C(\star)}{\Gamma \vdash \mathsf{ind}_{\mathbf{1}}^{C,c,\star} \equiv c : C(\star)}\ \mathbf{1}\text{-COMP}$$

El principio de inducción para $\mathbf{1}$ es el siguiente.

Teorema 0.7.4. *Existe una función*

$$\mathsf{ind}_{\mathbf{1}} : \prod_{C:\mathbf{1}\to\mathcal{U}} C(\star) \to \prod_{x:\mathbf{1}} C(x)$$

tal que

$$\mathsf{ind}_{\mathbf{1}}(C, c, \star) :\equiv c.$$

Demostración. La función definida por

$$\mathsf{ind}_{\mathbf{1}}(C, c, a) :\equiv \mathsf{ind}_{\mathbf{1}}^{C,c,a}$$

cumple los requisitos □

El principio de inducción del **1** nos permite usar la siguiente notación.

Notación 0.7.5. (Búsqueda de patrones para **1**)
Sea $C : \mathbf{1} \to \mathcal{U}$ una familia de tipos sobre **1**. Podemos definir una función $f : \prod_{(x:\mathbf{1})} C(x)$ por

$$f(\star) :\equiv \Phi$$

donde $\Phi : C(\star)$.

Justificación. Dado $\Phi : C(\star)$, podemos definir

$$f :\equiv \mathsf{ind}_{\mathbf{1}}(C, \Phi) : \prod_{x:\mathbf{1}} C(x)$$

y este cumple que $f(\star) \equiv \Phi$. □

Categóricamente, el tipo **1** es un objeto terminal de Type, pues dado un tipo C arbitrario, podemos definir una función $!\mathbf{1}_C : C \to \mathbf{1}$ por

$$!\mathbf{1}_C :\equiv \lambda(x : C).\star$$

Verificaremos que esta función es única en la Proposición **??**.

El tipo 2 y el tipo del coproducto

Podríamos ahora esperar poder definir **2**, el tipo con dos elementos, como la unión disjunta de **1** consigo mismo, a través de una un producto $\mathbf{2} :\equiv X \times \mathbf{1}$, donde X tiene dos elementos. Esto es posible, pero no tenemos ningún X que satisfaga esta propiedad.

Una alternativa es usando la noción del coproducto. Dados dos tipos A y B, su coproducto es denotado por $A + B$. Este puede ser entendido como una unión disjunta de los dos tipos.

Reglas 0.7.6. (Reglas de formación del coproducto.)

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

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{inl}_{A+B}(a) : A + B}\ +\text{-INTRO}_1$$

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash B : \mathcal{U}_i \qquad \Gamma \vdash b : B}{\Gamma \vdash \mathsf{inr}_{A+B}(b) : A + B}\ +\text{-INTRO}_2$$

De esta forma, inl_{A+B} y inr_{A+B} actúan como las inyecciones naturales hacia el coproducto. Así, podemos definir $\mathbf{2} :\equiv \mathbf{1} + \mathbf{1}$, y este posee dos elementos $\mathsf{inl}(\star)$ e $\mathsf{inr}(\star)$.

Sin embargo, para poder utilizar un coproducto, debemos saber cómo definir funciones cuyo dominio sean estos.

Reglas 0.7.7. (Reglas de eliminación del coproducto.)

$$\frac{\begin{array}{c}\Gamma \vdash C : A + B \to \mathcal{U}_i \qquad \Gamma \vdash c : \prod_{(x:A)} C(\mathsf{inl}(x)) \\ \Gamma \vdash d : \prod_{(y:B)} C(\mathsf{inr}(y)) \qquad \Gamma \vdash e : A + B\end{array}}{\Gamma \vdash \mathsf{ind}_{A+B}^{C,c,d,e} : C(e)}\ +\text{-ELIM}$$

$$\frac{\begin{array}{c}\Gamma \vdash C : A + B \to \mathcal{U}_i \qquad \Gamma \vdash c : \prod_{(x:A)} C(\mathsf{inl}(x)) \\ \Gamma \vdash d : \prod_{(y:B)} C(\mathsf{inr}(y)) \qquad \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{ind}_{A+B}^{C,c,d,\mathsf{inl}(a)} \equiv c(a) : C(\mathsf{inl}(a))}\ +\text{-COMP}_1$$

$$\frac{\begin{array}{c}\Gamma \vdash C : A + B \to \mathcal{U}_i \qquad \Gamma \vdash c : \prod_{(x:A)} C(\mathsf{inl}(x)) \\ \Gamma \vdash d : \prod_{(y:B)} C(\mathsf{inr}(y)) \qquad \Gamma \vdash b : B\end{array}}{\Gamma \vdash \mathsf{ind}_{A+B}^{C,c,d,\mathsf{inr}(b)} \equiv d(b) : C(\mathsf{inr}(b))}\ +\text{-COMP}_2$$

Estas reglas justifican el hecho de que basta definir una función en A y en B para que esté definida en $A + B$. Esto es el principio de inducción para el coproducto.

Teorema 0.7.8. *Sean A y B tipos. Entonces existe una función*

$$\mathsf{ind}_{A+B} : \prod_{(C:(A+B)\to\mathcal{U})} \left(\prod_{(x:A)} C(\mathsf{inl}(x))\right) \to \left(\prod_{(y:B)} C(\mathsf{inr}(y))\right) \to \prod_{(e:A+B)} C(e)$$

tal que

$$\mathsf{ind}_{A+B}(C, c, d, \mathsf{inl}(a)) \equiv c(a) : C(\mathsf{inl}(a))$$
$$\mathsf{ind}_{A+B}(C, c, d, \mathsf{inr}(b)) \equiv d(b) : C(\mathsf{inl}(b))$$

Demostración. La función definida por

$$\mathsf{ind}_{A+B}(C, c, d, e) :\equiv \mathsf{ind}_{A+B}^{C,c,d,e}$$

satisface los requisitos. □

Como en los tipos previos, podemos simplificar considerablemente la notación realizando una búsqueda de patrones.

Notación 0.7.9. (Búsqueda de patrones para el coproducto)
Sean A, B tipos, y $C : A + B \to \mathcal{U}$ una familia de tipos sobre $A + B$. Podemos definir una función $f : \prod_{(e:A+B)} C(e)$ por

$$f(\mathsf{inl}(a)) :\equiv \Phi_0$$
$$f(\mathsf{inr}(b)) :\equiv \Phi_1$$

donde $\Phi_0 : C(\mathsf{inl}(a))$ y $\Phi_1 : C(\mathsf{inr}(b))$ son expresiones que pueden tener a 'a' o a 'b' respectivamente.

Justificación. Dados $\Phi_0 : C(\mathsf{inl}(a))$ y $\Phi_1 : C(\mathsf{inr}(b))$, podemos definir

$$f :\equiv \mathsf{ind}_{A+B}(C, \lambda(a : A).\, \Phi_0, \lambda(b : B).\, \Phi_1) : \prod_{e:A+B} C(e)$$

y esta satisface $f(\mathsf{inl}(a)) :\equiv \Phi_0$ y $f(\mathsf{inr}(b)) :\equiv \Phi_1$. □

Analizaremos con más detalle el tipo $\mathbf{2} :\equiv \mathbf{1} + \mathbf{1}$. Escribiendo $0_{\mathbf{2}} :\equiv \mathsf{inl}(\star)$ y $1_{\mathbf{2}} :\equiv \mathsf{inr}(\star)$, y por el principio de inducción para el coproducto, vemos que para definir una función $\prod_{(x:\mathbf{2})} C(x)$ es suficiente definir dos funciones $f : \mathbf{1} \to C(0_{\mathbf{2}})$ y $g : \mathbf{1} \to C(1_{\mathbf{2}})$. Pero por el principio de inducción de $\mathbf{1}$, esto es lo mismo que seleccionar dos elementos, uno de $C(0_{\mathbf{2}})$ y otro de $C(1_{\mathbf{2}})$. De esta forma, obtenemos el principio de inducción para $\mathbf{2}$.

Teorema 0.7.10. *Sea $C : \mathbf{2} \to \mathcal{U}$ una familia de tipos sobre $\mathbf{2}$. Entonces existe una función*

$$\mathsf{ind}_{\mathbf{2}} : \prod_{(C:\mathbf{2}\to\mathcal{U})} C(0_{\mathbf{2}}) \to C(1_{\mathbf{2}}) \to \prod_{(x:\mathbf{2})} C(x)$$

tal que

$$\mathsf{ind}_{\mathbf{2}}(C, c_0, c_1, 0_{\mathbf{2}}) \equiv c_0 : C(0_{\mathbf{2}})$$
$$\mathsf{ind}_{\mathbf{2}}(C, c_0, c_1, 1_{\mathbf{2}}) \equiv c_1 : C(1_{\mathbf{2}})$$

Sea $C : \mathbf{2} \to \mathcal{U}$ definido por $C(0_{\mathbf{2}}) :\equiv A$ y $C(1_{\mathbf{2}}) :\equiv B$, es intuitivo que $\sum_{(x:2)} C(x)$ sea equivalente, en cierto sentido, a $A + B$. Este es efectivamente el caso (ver Capítulo 2), y pudimos también haber introducido primero el tipo $\mathbf{2}$ y definir $A + B$ a partir de este. Cuál es introducido primero no causa ninguna diferencia en los resultados.

0.8. El tipo de los naturales

Hemos introducido varios principios de inducción para diferentes tipos. Veamos que para el tipo de los naturales, su principio de inducción en DTT coincide con el principio usal de MC. Intuitivamente, el tipo de los naturales consiste del conjunto $\{0, 1, 2, \dots\}$, pero nótese que, a excepción del 0, todo elemento m puede ser escrito como $n+1$ para algún n. Esto sugiere las siguientes reglas.

Reglas 0.8.1. (Reglas de formación de $\mathbb{N}$.)

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

$$\frac{\Gamma \text{ ctx}}{\Gamma \vdash 0 : \mathbb{N}}\ \mathbb{N}\text{-INTRO}_1 \qquad \frac{\Gamma \text{ ctx}}{\Gamma \vdash \mathsf{succ} : \mathbb{N} \to \mathbb{N}}\ \mathbb{N}\text{-INTRO}_2$$

Aquí la función succ representa la función sucesor usual. Ahora, para formar funciones que tengan como dominio el tipo $\mathbb{N}$, necesitamos introducir las reglas de eliminación.

Reglas 0.8.2. (Reglas de eliminación de $\mathbb{N}$.)

$$\frac{\begin{array}{c}\Gamma \vdash C : \mathbb{N} \to \mathcal{U}_i \qquad \Gamma \vdash c_0 : C(0) \\ \Gamma \vdash c_s : \prod_{(n:\mathbb{N})} C(n) \to C(\mathsf{succ}(n)) \qquad \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}^{C,c_0,c_s,n} : C(n)}\ \mathbb{N}\text{-ELIM}$$

$$\frac{\begin{array}{c}\Gamma \vdash C : \mathbb{N} \to \mathcal{U}_i \qquad \Gamma \vdash c_0 : C(0) \\ \Gamma \vdash c_s : \prod_{(n:\mathbb{N})} C(n) \to C(\mathsf{succ}(n))\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}^{C,c_0,c_s,0} \equiv c_0 : C(n)}\ \mathbb{N}\text{-COMP}_1$$

$$\frac{\begin{array}{c}\Gamma \vdash C : \mathbb{N} \to \mathcal{U}_i \qquad \Gamma \vdash c_0 : C(0) \\ \Gamma \vdash c_s : \prod_{(n:\mathbb{N})} C(n) \to C(\mathsf{succ}(n)) \qquad \Gamma \vdash n : \mathbb{N}\end{array}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}^{C,c_0,c_s,\mathsf{succ}(n)} \equiv c_s(n, \mathsf{ind}_{\mathbb{N}}^{C,c_0,c_s,n}) : C(\mathsf{succ}(n))}\ \mathbb{N}\text{-COMP}_2$$

Aquí, nuevamente, $\mathsf{ind}_{\mathbb{N}}^{C,c_0,c_s,\mathsf{succ}(n)}$ es un primitivo que pertenece a $C(n)$. Con estas reglas, obtenemos el principio de inducción usual.

Teorema 0.8.3. *Existe una función*

$$\mathsf{ind}_{\mathbb{N}} : \prod_{(C:\mathbb{N}\to\mathcal{U})} C(0) \to \Big(\prod_{(n:\mathbb{N})} C(n) \to C(\mathsf{succ}(n))\Big) \to \prod_{(n:\mathbb{N})} C(n)$$

tal que

$$\begin{aligned}\mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, 0) &:\equiv c_0, \\ \mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, \mathsf{succ}(n)) &:\equiv c_s(n, \mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, n)).\end{aligned}$$

Demostración. La función definida por

$$\mathsf{ind}_{\mathbb{N}}(C, c_0, c_s, n) :\equiv \mathsf{ind}_{\mathbb{N}}^{C, c_0, c_s, n}$$

cumple los requisitos. □

Notación 0.8.4. (Búsqueda de patrones para los naturales)
Podemos definir una función con dominio $\mathbb{N}$ a una familia de tipos $C : \mathbb{N} \to \mathcal{U}$ definiendo su comportamiento en 0 y $\mathsf{succ}(n)$, es decir

$$f(0) :\equiv \Phi_0 : C(0)$$
$$f(\mathsf{succ}(n)) :\equiv \Phi_1 : C(\mathsf{succ}(n))$$

donde n y $f(n)$ pueden aparecer en Φ_1.

Justificación. f puede ser definida como

$$f :\equiv \mathsf{ind}_{\mathbb{N}}(C, \Phi_0, \lambda n.\, \lambda r.\, \Phi_1^*)$$

donde Φ_1^*, es Φ con todas las ocurrencias de $f(n)$ reemplazadas por r. □

Definimos los símbolos usuales para los números naturales como aplicaciones repetidas del constructo succ. Es decir $1 :\equiv \mathsf{succ}(0)$, $2 :\equiv \mathsf{succ}(1)$, $3 :\equiv \mathsf{succ}(2), \ldots$.

En el caso en el que la familia $C : \mathbb{N} \to \mathcal{U}$ es constante, el principio de inducción se convierte en el principio de recursión.

Teorema 0.8.5. *Existe una función*

$$\mathsf{rec}_{\mathbb{N}} : \prod_{C:\mathcal{U}} \mathbb{N} \to (\mathbb{N} \to \mathbb{N}) \to (\mathbb{N} \to C)$$

Demostración. Podemos definir $\mathsf{rec}_{\mathbb{N}}$ por

$$\mathsf{rec}_{\mathbb{N}}(C, c, f, 0) :\equiv c : C$$
$$\mathsf{rec}_{\mathbb{N}}(C, c, f, \mathsf{succ}(n)) :\equiv f(\mathsf{rec}_{\mathbb{N}}(C, c, f, n)) : C$$

□

Es más, para todos los tipos existe un principio de recursión, el cual es simplemente el principio de inducción aplicado a una familia constante. Veamos dos aplicaciones de esto.

Ejemplo 0.8.6. Existe una función $\mathsf{double} : \mathbb{N} \to \mathbb{N}$ tal que duplica el valor su input.

$$\mathsf{double} : \mathbb{N} \to \mathbb{N}$$
$$\mathsf{double}(0) :\equiv 0$$
$$\mathsf{double}(\mathsf{succ}(n)) :\equiv \mathsf{succ}(\mathsf{succ}(\mathsf{double}(n)))$$

Por ejemplo,

$$\begin{aligned}\mathsf{double}(2) &\equiv \mathsf{double}(\mathsf{succ}(\mathsf{succ}(0)))\\ &\equiv \mathsf{succ}(\mathsf{succ}(\mathsf{double}(\mathsf{succ}(0))))\\ &\equiv \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathsf{double}(0)))))\\ &\equiv \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(0))))\\ &\equiv 4\end{aligned}$$

La función suma también es definida por recursión

Ejemplo 0.8.7.

$$\begin{aligned}\mathsf{add} &: \mathbb{N} \to \mathbb{N} \to \mathbb{N}\\ \mathsf{add}(0) &:\equiv \mathsf{id}_{\mathbb{N}}\\ \mathsf{add}(\mathsf{succ}(n)) &:\equiv \lambda m.\, \mathsf{succ}(\mathsf{add}(n)(m))\end{aligned}$$

A forma de recordatorio que la notación previa es solo una simplificación, mostramos la definición usando solo el principio de inducción:

$$\mathsf{add} :\equiv \mathsf{ind}_{\mathbb{N}}(\lambda n.\, (\mathbb{N} \to \mathbb{N}), \mathsf{id}_{\mathbb{N}}, \lambda n.\, \lambda g.\, \lambda m.\, \mathsf{succ}(g(m)))$$

Como es común, utilizaremos $a + b$ para referirnos a $\mathsf{add}(a)(b)$.

0.9. Proposiciones como tipos

En la introducción habíamos mencionado que las proposiciones eran representadas como tipos en DTT. Comenzaremos analizando cada uno de los constructos lógicos principales en MC.

- Para mostrar que se cumple $P \wedge Q$, es suficiente mostrar que se cumplen ambos P y Q. Podemos entonces considerar una prueba de $P \wedge Q$ como un par de pruebas de P y Q.
- Para mostrar que se cumple $P \vee Q$, es suficiente mostrar que se cumple alguno de P o Q. Podemos entonces considerar una prueba de $P \vee Q$ como la unión disjunta de pruebas de P y Q.
- Para mostrar que se cumple $P \implies Q$, es suficiente mostrar que dado P, se puede mostrar que se cumple Q. Podemos entonces considerar una prueba de $P \implies Q$ como una función que toma una prueba de P y devuelve una prueba de Q.
- Para mostrar que se cumple $\neg P$, es suficiente mostrar que si es que tenemos P, podemos llegar a una contradicción. Podemos entonces considerar una prueba de $\neg P$ como una función que toma una prueba de P y devuelve una prueba de una contradicción.
- Para mostrar que se cumple $\forall x \in A, P(x)$, es suficiente mostrar que, dado un x arbitrario en A, se puede mostrar que este satisface $P(x)$. Podemos entonces considerar una prueba de $\forall x \in A, P(x)$ como una función que toma cualquier elemento $x \in A$ y devuelve una prueba de $P(x)$.

- Para mostrar que se cumple $\exists x \in A, P(x)$, es suficiente mostrar que existe un $a \in A$ tal que $P(a)$. Podemos entonces considerar una prueba de $\exists x \in A, P(x)$ como un par, el cual consiste de un elemento $a \in A$ y una prueba de $P(a)$.

La interpretación de estos constructos de esta manera es llamada la **interpretación BHK**, debido a los nombres de sus principales proponentes L. E. J. Brouwer, Arend Heyting, y Andrey Kolmogorov. Notemos su similitud con las reglas de introducción para los tipos que ya hemos introducido.

Por ejemplo, la regla de introducción para el tipo del producto indica que para construir un elemento de $P \times Q$ es suficiente mostrar dos elementos $a : P$ y $b : Q$, y el nuevo elemento construido sería el par (a, b) de estos. Así, los tipos se interpretan como proposiciones y los elementos de estos tipos se interpretan como pruebas o evidencia de estas proposiciones. Esta correspondencia entre lógica y tipos es llamada el **isomorfismo de Curry-Howard**.

Matemática Clásica	Teoría de Tipos dependientes
$A \wedge B$	$A \times B$
$A \vee B$	$A + B$
$A \implies B$	$A \to B$
$A \iff B$	$(A \to B) \times (B \to A)$
$\neg A$	$A \to \mathbf{0}$
$\forall x \in A, P(x)$	$\prod_{(x:A)} P(x)$
$\exists x \in A, P(x)$	$\sum_{(x:A)} P(x)$

Cuadro 2: Isomorfismo de Curry-Howard.

La correspondencia es relativamente simple por lo que no la explicaremos más a detalle, solo el caso de $\neg A$ merece una mayor atención. En DTT el $\mathbf{0}$ toma el rol de un absurdo, puesto que su mismo principio de inducción indica que es posible formar una función con cualquier codominio dado un elemento de $\mathbf{0}$. Por este motivo, interpretamos $A \to \mathbf{0}$ como $\neg A$.

Definición 0.9.1. Diremos que dos tipos A y B son lógicamente equivalentes cuando se tiene $A \iff B$; es decir, tenemos dos funciones $f : A \to B$ y $g : B \to A$, como indicado en el Cuadro 2.

Veamos algunos ejemplos de lógica proposicional y lógica predicativa.

Ejemplo 0.9.2. Una de las leyes de Morgan indica que para todo par de proposiciones A y B tenemos

$$\neg A \vee \neg B \implies \neg(A \wedge B).$$

El equivalente en DTT es la existencia de una función

$$f : \prod_{(A:\mathcal{U})} \prod_{(B:\mathcal{U})} (A \to \mathbf{0}) + (B \to \mathbf{0}) \to (A \times B \to \mathbf{0})$$

En efecto, combinando la búsqueda de patrones en el coproducto y en el producto, podemos definir f como

$$f(A, B, \mathsf{inl}(f_1), (a, b)) :\equiv f_1(a)$$
$$f(A, B, \mathsf{inr}(f_2), (a, b)) :\equiv f_2(b)$$

Notación 0.9.3. A veces omitiremos la mención explícita de los Π tipos involucrados, cuando estos toman el rol lógico de "para todo", entendiendo que el término que estamos construyendo es una función dependiente con un dominio apropiado.

Ejemplo 0.9.4. Consideremos la siguiente implicación usada comúnmente

$$\exists x \in A, \neg P(x) \implies \neg(\forall x \in A, P(x))$$

Esta es representada por la existencia de una función

$$f : \textstyle\sum_{(x:A)}(P(x) \to \mathbf{0}) \to (\prod_{(x:A)} P(x)) \to \mathbf{0}$$

Esta puede ser definida por

$$f((a, g), h) :\equiv g(h(a))$$

Ejemplo 0.9.5. El axioma de elección en MC indica que dada una colección de conjuntos no vacíos X, es posible crear una función $f : X \to \bigcup X$ que selecciona un elemento de cada conjunto. Es decir,

$$\left(\forall x \in X, \exists y \in \bigcup x, (y \in x)\right) \implies \left(\exists f : X \to \bigcup X, \forall x \in X, (f(x) \in x)\right)$$

En DTT, esto podría ser entendido[3] como un caso particular de

$$\mathsf{ac} : \left(\textstyle\prod_{(x:A)}\sum_{(y:B)} R(x, y)\right) \to \left(\sum_{(f:A\to B)}\prod_{(x:A)} R(x, f(x))\right)$$

Podemos definir esta función como

$$\mathsf{ac}(h) :\equiv \left(\lambda x.\, \mathsf{pr}_1(h(x)), \lambda x.\, \mathsf{pr}_2(h(x))\right)$$

Esta función está bien definida puesto que

$$\lambda x.\, \mathsf{pr}_1(g(x)) : A \to B,$$
$$\lambda x.\, \mathsf{pr}_2(g(x)) : \textstyle\prod_{(x:A)} R(x, \mathsf{pr}_1(g(x))).$$

como es requerido por el tipo del codomino de ac.

Definición 0.9.6. Dado un tipo A, si existe un elemento de él tal que $a : A$, diremos que A es un tipo **habitado**.

Así, se dice una proposición P es **verdadera** cuando es un tipo habitado. El ejemplo previo muestra que el axioma de elección es verdadero en esta teoría.

[3] La verdadera formalización del axioma de elección como visto de manera clásica, es ligeramente distinta a la presentación actual, ver [17, Sección 3.8].

0.10. El tipo de identidades

De acuerdo a la interpretación de proposiciones como tipos, debería haber un tipo que refleje la propiedad de que dos elementos sean iguales. Es decir, si tenemos $x, y : A$, tenemos un tipo de identidades entre x y y, el cual denotamos por $x =_A y$.

Reglas 0.10.1. (Reglas de formación del tipo de identidades.)

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b : \mathcal{U}_i} \text{ =-FORM}$$

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}_a : a =_A a} \text{ =-INTRO}$$

Intuitivamente, el tipo $x =_A y$ representa la noción de igualdad, y este está habitado justamente cuando x es igual a y. En particular, $x =_A x$ siempre está habitado, puesto que por =-INTRO tenemos una función

$$\mathsf{refl} : \prod_{x:A} x =_A x,$$

definida por $\mathsf{refl}(x) :\equiv \mathsf{refl}_x$. Además, notemos que es imposible formar el tipo de igualdades para dos elementos en dos tipos distintos.

La introducción de este tipo permite enunciar y probar teoremas que involucren la igualdad.

Ejemplo 0.10.2. Por definición, para todo $n : \mathbb{N}$ tenemos $0 + n :\equiv n$, por lo que podemos definir

$$\lambda n.\, \mathsf{refl}_n : \prod_{n:\mathbb{N}} (0 + n =_{\mathbb{N}} n)$$

Por otro lado, la demostración de $\prod_{(n:\mathbb{N})}(n + 0 =_{\mathbb{N}} n)$ requiere de otros resultados, como el principio de inducción para el tipo de identidades.

De manera análoga a los casos de la suma dependiente y el coproducto, dado que la forma canónica de introducir un elemento del tipo de identidades es por refl_x, basta definir una función dependiente en estos elementos para que esté definida en todo $x =_A y$. Este razonamiento nos da:

Reglas 0.10.3. (Reglas de eliminación del tipo de identidades.)

$$\frac{\begin{array}{c}\Gamma \vdash C : \prod_{(x,y:A)}(x =_A y) \to \mathcal{U}_i \qquad \Gamma \vdash c : \prod_{(z:A)} C(z, z, \mathsf{refl}_z) \\ \Gamma \vdash a : A \qquad \Gamma \vdash b : A \qquad \Gamma \vdash p : a =_A b\end{array}}{\Gamma \vdash \mathsf{ind}_{=_A}^{C,c,a,b,p} : C(a, b, p)} \text{ =-ELIM}$$

$$\frac{\begin{array}{c}\Gamma \vdash C : \prod_{(x,y:A)}(x =_A y) \to \mathcal{U}_i \qquad \Gamma \vdash c : \prod_{(z:A)} C(z, z, \mathsf{refl}_z) \\ \Gamma \vdash a : A\end{array}}{\Gamma \vdash \mathsf{ind}_{=_A}^{C,c,a,a,\mathsf{refl}_a} : C(a, a, \mathsf{refl}_a)} \text{ =-COMP}$$

Como en los tipos previos, $\mathsf{ind}_{=_A}^{C,c,a,a,p}$ es un primitivo que existe bajo ciertas condiciones, y estas reglas nos dan un principio de introducción y una búsqueda de patrones.

Teorema 0.10.4. *Para todo tipo $A : \mathcal{U}$, existe una función*

$$\mathsf{ind}_{=_A} : \prod_{(C:\prod_{(x,y:A)}(x=_A y)\to\mathcal{U})} \left(\prod_{(x:A)} C(x,x,\mathsf{refl}_x)\right) \to \prod_{(x,y:A)} \prod_{(p:x=_A y)} C(x,y,p)$$

tal que

$$\mathsf{ind}_{=_A}(C,c,x,x,\mathsf{refl}_x) :\equiv c(x).$$

Demostración. Podemos definir $\mathsf{ind}_{=_A}$ por

$$\mathsf{ind}_{=_A}(C,c,x,x,p) :\equiv \mathsf{ind}_{=_A}^{C,c,a,a,p}.$$

□

Notación 0.10.5. (Búsqueda de patrones para el tipo de identidades)
Podemos definir una función $f : \prod_{(x,y:A)} \prod_{(p:x=_A y)} C(x,y,p)$ definiendo su comportamiento en refl_x, es decir

$$f(\mathsf{refl}_x) :\equiv \Phi : C(x,x,\mathsf{refl}_x)$$

Justificación. La función f puede ser definida como

$$f :\equiv \mathsf{ind}_{=_A}(C, \lambda x.\, \Phi)$$

□

Así, para definir una función $f : \prod_{(x,y:A)} \prod_{(p:x=_A y)} C(x,y,p)$ basta asumir que x es y, y definir el comportamiento de la función en los caminos refl_x. Utilizaremos esto múltiples veces en los siguientes capítulos.

El tipo de identidades es quizás el que mayor riqueza trae a esta teoría, y el siguiente capítulo estará dedicado principalmente a este.

Notación 0.10.6. Escribiremos $x = y$ en vez de $x =_A y$ cuando no haya riesgo de confusión.

Para diferenciar el tipo $x =_A y$ del juicio $x \equiv y$, a veces se llama al primero de estos como una **igualdad proposicional**.

0.11. Grupos

Veamos como los tipos introducidos nos permiten formalizar en DTT el concepto de un grupo. Recordemos que en MC un grupo es par $(G, *)$, tal que G es un conjunto y $*$ es una operación binaria en G tal que

- $*$ es asociativa,

- existe un elemento neutro e, tal que para todo $g \in G$ se tiene que $e * g = g * e = g$, y
- para todo $g \in G$ existe un $g^{-1} \in G$ tal que $g * g^{-1} = g^{-1} * g = e$.

Esta estructura puede ser reflejada directamente en la siguiente definición

Definición 0.11.1. Dado un tipo A, diremos que este tiene una **estructura de grupo** si es que el tipo

$$\begin{aligned}\mathsf{GroupStr}(A) :\equiv \sum_{m:A\to A\to A} \prod_{x,y,z:A} (m(x,m(y,z)) = m(m(x,y),z)) \\ \times \sum_{e:A} (e*g=g)\times(g*e=g) \\ \times \sum_{i:A\to A} (g*i(g)=e)\times(i(g)*g=e)\end{aligned}$$

está habitado.

Cada línea de la definición previa corresponde a uno de los puntos de la definición clásica. Con el concepto de estructura ya definido, podemos definir lo que es un grupo.

Definición 0.11.2. Un **grupo** es un tipo junto con una estructura de grupo. Es decir, un elemento del tipo

$$\mathsf{Group} :\equiv \sum_{A:\mathcal{U}} \mathsf{GroupStr}(A)$$

Nótese que, a diferencia de MC, sí tenemos un tipo de todos los grupos, el cual es Group.

Es claro que una formalización similar aplica para la gran mayoría de teorías algebraicas, como las de los anillos, módulos, categorías, etc.

0.12. Metateoría de DTT

Hemos introducido las reglas de DTT que usaremos en lo que sigue del presente documento. En la presente sección discutiremos sobre las reglas mismas, y sobre las propiedades de la teoría que estas generan; es decir, la *metateoría*.

En primer lugar, parecería que existe cierta inconsistencia en algunas reglas. Por ejemplo, consideremos la siguiente regla ya introducida

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

Se tiene $\Gamma \vdash a : A$ en la hipótesis; sin embargo, no estamos afirmando que A sea un tipo, por lo que podría ser posible que $a : A$ sea una expresión sin sentido.

Esta inquietud, aunque válida, no está justificada, pues tenemos el siguiente resultado.

Teorema 0.12.1. *Si se tiene* $\Gamma \vdash a : A$ *para algún* Γ*, también se tiene que* $\Gamma \vdash A : \mathcal{U}_i$.

Nótese que a diferencia de los resultados y ejemplos previos, este resultado es un resultado sobre la teoría misma. Los teoremas de incompletitud de Gödel [7] están en este mismo nivel.

Demostración. La técnica a través la cual se muestran este tipo de teoremas en DTT es a través de *inducción estructural*. Recordemos que una derivación de un juicio es en realidad un árbol invertido con el juicio por derivar en la raíz del árbol. Por lo tanto, podemos utilizar inducción en el árbol, de acuerdo a la última regla usada para llegar a $\Gamma \vdash a : A$.

Por ejemplo, si la última regla usada fue $\mathbb{N}$-INTRO$_1$, tenemos que $\Gamma \vdash 0 : \mathbb{N}$, y por $\mathbb{N}$-FORM sabemos que $\mathbb{N}$ es un tipo. Un razonamiento análogo por casos para cada regla relevante concluye la demostración. □

Otros dos resultados importantes, que ya hemos utilizado libremente en los ejemplos son:

Teorema 0.12.2. *(Substitución) Si se tiene* $\Gamma \vdash a : A$ *y* $\Gamma, x : A, \Delta \vdash b : B$*, entonces es posible derivar* $\Gamma, \Delta[a/x] \vdash b[a/x] : B[a/x]$.

Teorema 0.12.3. *(Debilitación) Si se tiene* $\Gamma \vdash A : \mathcal{U}_i$ *y* $\Gamma, \Delta \vdash b : B$*, entonces es posible derivar* $\Gamma, x : A, \Delta \vdash b : B$.

La demostración de estos teoremas para teorías similares se puede encontrar en [12], por ejemplo.

El último metateorema que presentaremos sobre DTT es que esta teoría es consistente, es decir no se puede derivar una contradicción a partir de las reglas ya introducidas.

Teorema 0.12.4. *(Consistencia de DTT) No es posible derivar una contradicción en DTT, es decir* $\cdot \vdash x : \mathbf{0}$ *para algún* x*; asumiendo que la teoría usual de MC también es consistente.*

Omitimos también la demostración de este resultado, que se puede encontrar en [10].

0.13. Comentarios adicionales

La complejidad de DTT

La introducción de las reglas y la derivación de juicios podría hacer parecer que DTT es una teoría mucho más complicada que la lógica clásica; sin embargo, este no es el caso. La lógica clásica en realidad también está formalizada de una forma similar, por ejemplo se tiene la siguiente regla para la introducción de la conjunción:

$$\frac{\varphi \qquad \psi}{\varphi \wedge \psi} \wedge\text{-INTRO}$$

Así existen reglas para cada constructo lógico ($\forall, \exists, \wedge, \vee, \neg, \implies$), y resultados análogos a los de la sección previa [2], los cuales son asumidos implicitamente por los matemáticos, razón por lo que omitimos las para DTT.

La diferencia es entonces que las reglas de la lógica clásica son usualmente introducidas de manera informal, mientras que hemos decidido presentar las reglas de DTT formalmente en este trabajo. Diversas presentaciones de DTT introducen los constructos de forma informal, como [11] y [6].

Constructivismo

La lógica subyacente en DTT es una lógica **constructiva**, también llamada **lógica intuicionista**. Por ejemplo, en esta no es posible demostrar $P \vee \neg P$, la ley del medio excluido. Es posible añadir esta ley como una regla adicional[4], pero esto no será necesario en el desarrollo actual.

Ante cierta inspección, es fácil notar que esta ley es en cierto sentido problemática. Esta permite afirmar la existencia de objetos que no han sido construidos explícitamente, o simplificar problemas de una forma no obvia. Por ejemplo, consideremos la siguiente proposición:

Proposición 0.13.1. *Existe un número natural n, tal que $n = 0$ si y solo si la hipótesis de Riemann es verdadera.*

Demostración. Si la hipótesis de Riemann es verdadera, entonces n es 0. Si la hipótesis de Riemann es falsa, entonces podemos poner $n = 1$. □

Este número n refleja la veracidad de la hipótesis de Riemann de una forma no obvia. Por otro lado, si la prueba fuese constructiva, habríamos generado un n específico, y podríamos verificar fácilmente si este es o no diferente de 0.

No asumir la ley del medio excluido significa que todos los elementos de los cuales tenemos información has sido construidos paso a paso, por lo que además de su existencia, podemos indagar sobre otras propiedades relevantes que pueden tener.

Otro principio relacionado es el de reducción al absurdo, $\neg\neg P \implies P$. Este es en realidad equivalente a la ley del medio excluido, y permite probar la existencia de elementos sin saber cuáles son. Por ejemplo:

Proposición 0.13.2. *En la expansión decimal de π, existen dos dígitos que se repiten infinitas veces.*

Demostración. Si no fuese así, se tendría que un solo dígito se repite infinitas veces, o que ningún dígito se repite infinitas veces. En ambos casos, concluimos que π es un racional, lo cual es un absurdo. □

[4]Esto es posible en DTT, pero una vez introducido el axioma de univalencia esto se vuelve inconsistente. En cambio, se puede introducir un axioma similar, que solo involucra proposiciones simples (ver Sección **??**).

Sin embargo, en esta prueba no hemos hallado cuáles son estos dos dígitos; es decir, la prueba no es constructiva.

Esto no quiere decir que no podamos utilizar nunca el razonamiento de $P \vee \neg P$, solo que para utilizarlo es necesario primeramente demostrar que esto se cumple para el tipo P en cuestión. Realizaremos esto para las igualdades en los naturales en la Sección **??**.

Asistentes de pruebas

Incluso grandes matemáticos como Leibniz, Gauss, Andrew Wiles, entre otros, han cometido graves errores en sus demostraciones, y ellos no fueron los primeros ni serán los últimos. A fin de evitar esto, existen los llamados **asistentes de pruebas**, programas que verifican que una demostración es correcta. La mayoría de estos usan DTT en vez de teoría de conjuntos, consideremos el siguiente ejemplo tomado de [1] para ver por qué:

Proposición 0.13.3. *Sean U y V espacios vectoriales y $f : U \to V$ una función lineal. Entonces $f(2x + y) = 2f(x) + f(y)$.*

La proposición es fácilmente entendida por un lector que conozca estos conceptos, y pareciese que se ha presentado con precisión los variables necesarias, pero veamos la gran cantidad de información omitida:

- Se ha entendido que existe un campo K subyacente a U y a V.
- Se ha entendido que f es en realidad una función entre los conjuntos subyacentes $f : |U| \to |V|$, recordemos que un espacio vectorial U es un triple $(|U|, \cdot, +)$.
- Se ha entendido que x y y son elementos arbitrarios de $|U|$.
- Se ha entendido que el '$+$' en la izquierda de la ecuación es la suma asociada a U, mientras que el '$+$' en la derecha es el asociado a V.
- Finalmente, se ha entendido que 2 es $1 + 1$, donde 1 es el neutro de la multiplicación del campo K.

Esta gran cantidad de información no puede ser inferida correctamente por computadoras que formalicen la teoría de conjuntos. Uno de los principales problemas es que la estructura que tiene cierto constructo, como los espacios vectoriales, no es reflejada de una manera única en teoría de conjuntos. Por poner otro ejemplo, dada la construcción de los reales como cortes de Dedekind, no solo tiene sentido la proposición '$0_{\mathbb{Q}} \in 1_{\mathbb{R}}$'; peor aún, es verdadera.

En contraste, en DTT, los elementos de un tipo pertenecen a un único tipo (a excepción de los tipos mismos), y la estructura adicional es parte esencial y única de una definición (ver 0.11.2, por ejemplo). Estos hechos permiten la inferencia del significado de variables no completamente especificadas, en la gran mayoría de los casos. Esto hace que DTT sea una teoría más apropiada para la formalización de las matemáticas, a través de programas de computadoras.

Ya existe un gran esfuerzo a nivel global de traducir la matemática clásica al lenguaje de DTT, usado por los asistentes de pruebas, y así permitir la

verificación de estos resultados por computadora. Entre estos, mencionamos las formalizaciones [9] y [16], proyectos con más de 250 contribuidores en conjunto.

Gran parte de resultados que consideramos básicos, y que se enseñan a nivel de pregrado ya están formalizados. Sin embargo, también se han verificado algunos resultados más modernos y complicados. El teorema de Feit-Thompson [5], que indica que todo grupo de orden finito e impar es soluble, fue verificado usando el programa Coq en el 2013 [8].

Otro ejemplo es la verificación de un resultado avanzado de geometría algebraica de Peter Scholze, ganador del premio Fields 2018, en colaboración con Dustin Clausen. El proceso de formalización comenzó como un reto de Scholze a la comunidad de asistentes de pruebas [15]:

> Considero que este teorema es de una gran importancia fundacional, por lo que estar 99.9 % seguro no es suficiente… Pasé mucho del 2019 obsesionado con la prueba de este teorema, casi volviéndome loco sobre esta. Al final logramos escribir el argumento en un artículo, pero creo que nadie más se ha atrevido a ver los detalles de este, así que todavía tengo algunas pequeñas dudas.

Solo 6 meses después, con la ayuda de varios matemáticos y científicos de la computación, Scholze escribe que su reto era prácticamente un éxito: aunque todavía no se había demostrado el teorema, todos los lemas que causaban cierta duda ya estaban formalizados [14]. A través de este ejercicio, él comenta, no solo encontró múltiples errores (que afortunadamente fueron posibles de solucionar), también profundizó el entendimiento de su propia prueba.

Esta es la propuesta de los proponentes de los asistentes de pruebas: introducir a las computadoras como una herramienta más en el arsenal que posee un matemático, y obtener una mejor comprensión del tema de estudio, así como la posibilidad de no volver a equivocarnos nunca más[5].

El presente trabajo advoca esta posición, por lo que todos los resultados aquí descritos has sido formalizados en un asistente de pruebas, llamado Agda. El código se encuentra disponible en línea, a través de la siguiente página web interactiva https://shiranaiyo.github.io/MastersThesis/.

Enfatizamos que la posibilidad de la formalización en un asistente de pruebas es solo una ventaja más de DTT. Como mencionamos, la ventaja principal es el tratamiento uniforme de proposiciones y tipos, la cual será aprovechada en los siguientes capítulos.

[5]Podría preguntarse uno, ¿qué garantiza que el programa no cometa un error? La respuesta es: matemática. Se puede razonar de una forma similar a la de inducción estructural, y verificar que cada paso del programa es correcto, y por lo tanto, el algoritmo entero.