Introducción a la correspondencia entre pruebas y programas:

La teoría de tipos de Martin-Löf

Alexandre Miquel

mayo de 2021

Interpretación de Brouwer-Heyting-Kolmogorov

Filosofía del constructivismo: El significado de una proposición A es el conjunto $\Phi(A)$ de las "evidencias" (sentido intuitivo) que A se cumple:

$$
\begin{aligned}
\Phi(A \wedge B) &= \Phi(A) \times \Phi(B) && \text{(Producto cartesiano)} \\
\Phi(A \vee B) &= \Phi(A) + \Phi(B) && \text{(Suma directa)} \\
\Phi(A \Rightarrow B) &= \Phi(A) \to \Phi(B) && \text{(funciones "computables")} \\
\Phi(\bot) &= \varnothing && \text{(Conjunto vacío)} \\
\Phi(\top) &= \{\bullet\} && \text{(Conjunto unitario)} \\
\Phi(\forall x : D\,.\;\; A(x)) &= \prod_{x \in D} \Phi(A(x)) && \text{(Producto dependiente)} \\
\Phi(\exists x : D\,.\;\; A(x)) &= \sum_{x \in D} \Phi(A(x)) && \text{(Suma dependiente)}
\end{aligned}
$$

Ejemplo típico: $\forall x : \mathtt{Nat}\,.\;\; \exists y : \mathtt{Nat}\,.\;\; A(x, y)$

Historia (1/3)

1969 W. A. Howard: *The formulae-as-types notion of construction.* (Apuntes privados con difusión restringida)

1971 J.-Y. Girard: *Une extension de l'interprétation fonctionnelle de Gödel à l'analyse et son application à l'élimination des coupures dans l'analyse et la théorie des types* (Sistema F)

1971 P. Martin-Löf: *A theory of types* (Sistema «Type : Type»)

1972 J.-Y. Girard: *Interprétation fonctionnelle et élimination des coupures de l'arithmétique d'ordre supérieur* (Sistema Fω)

1975 P. Martin-Löf: *An Intuitionistic Theory of Types: Predicative Part* (Teoría de tipos intensional)

1984 P. Martin-Löf: *Intuitionistic Type Theory* (Teoría de tipos extensional)

1985 T. Coquand & G. Huet: *Constructions: A Higher Order Proof System for Mechanizing Mathematics* (Cálculo de construcciones)

1989 C. Paulin: *Le calcul des constructions inductives* (CIC, Coq)

Historia (2/3)

P. Martin-Löf: *A Theory of Types* (1971):

In what follows, mathematical objects will be regarded as our own constructions. Every mathematical object is of a certain kind or type which is uniquely associated with the object in question. A type is defined by prescribing how we are allowed to construct objects of that type. The types themselves are mathematical objects, namely, those objects whose type is the type of types. [. . .]

A proposition is defined by prescribing what we have to do in order to prove it. For example,

$$971 \text{ is a non prime number}$$

is a proposition which we prove by exhibiting two natural numbers greater than one and a computation which shows that their product equals 971*. The similarity between the notion of proposition and the notion of type described above is not accidental. Indeed, a proposition may always be regarded as a type, namely, the type of proofs of that proposition, and, conversely, a type always determines a proposition, namely, the proposition which we prove by exhibiting an object of that type. This explains why I shall treat the notion of type and the notion of proposition as one and the same notion, thereby taking seriously the analogy between types [...] and propositions discovered by [Curry and Feys 1958] in the case of the positive implicational calculus and extended to Heyting arithmetic by [Howard 1969].*

Historia (3/3)

P. Martin-Löf: *A Theory of Types* (1971):

En lo siguiente, los objetos matemáticos serán considerados como nuestras propias construcciones. Cada objeto matemático es de cierto género o tipo, asociado de modo único con dicho objeto. Un tipo está definido prescribiendo cómo uno puede construir objetos de ese tipo. Los tipos en sí mismos son objetos matemáticos, a saber, los objetos cuyo tipo es el tipo de los tipos. [. . .]

Una proposición está definida prescribiendo cómo uno puede demostrarla. Por ejemplo,

$$971 \text{ es un número no primo}$$

es una proposición que se demuestra exhibiendo dos enteros naturales mayores que uno y una computación que muestra que su producto vale 971*. La semejanza entre la noción de proposición y la noción de tipo descritas previamente no es accidental. En efecto, una proposición siempre puede ser vista como un tipo, es decir, el tipo de las demostraciones de esta proposición, y recíprocamente, un tipo siempre determina una proposición, es decir: la proposición que se demuestra exhibiendo un objeto de este tipo. Esto explica por qué voy a tratar la noción de tipo y la noción de proposición como una misma noción, considerando seriamente la analogía entre los tipos [. . .] y las proposiciones descubierta por [Curry and Feys 1958] en el caso del cálculo implicacional positivo y extendida a la Aritmética de Heyting por [Howard 1969].*

Ideas fundamentales de la teoría de tipos

- Tipo = «métodos» que permiten construir objetos de este tipo
- Proposición = «métodos» que permiten demostrar esta proposición
- Por lo tanto: Proposición = Tipo (identificación completa)
- En particular: $A \Rightarrow B = A \to B$, $A \wedge B = A \times B$ (etc.)
- Demostración = término de prueba
- Cada objeto matemático tiene un tipo (único)
- Los tipos también tienen sus propios tipos:

  Set, Type, ... = tipos de los tipos = universos

  (se necesita distinguir varios níveles de tipos para evitar las paradojas)
- Una nueva construcción: el producto dependiente

  $\Pi x : A \,.\, B(x) = \forall x : A \,.\, B(x)$ (véase más adelante)

Tipos no dependientes...

- Hasta ahora, sólo vimos tipos de base ($\texttt{Unit}$, $\texttt{Bool}$, $\texttt{Nat}$) o tipos construidos a partir de otros tipos ($A \to B$, $A \times B$, $A + B$)
- **Ejemplo:** El tipo de las listas

Tipos $\quad A, B \quad ::= \quad \cdots \quad | \quad \texttt{List}(A)$

- $[] \ : \ \texttt{List}(A)$ (para todo tipo A)
- $[\texttt{true};\ \texttt{true};\ \texttt{false};\ \texttt{true}] \ : \ \texttt{List}(\texttt{Bool})$
- $[18;\ 42;\ 7;\ 28;\ 0;\ 13] \ : \ \texttt{List}(\texttt{Nat})$
- $[\texttt{succ};\ (\lambda x : \texttt{Nat} \,.\, 0);\ \texttt{plus}\ 42] \ : \ \texttt{List}(\texttt{Nat} \to \texttt{Nat})$
- Sea la función $\texttt{make\_list}_A\ n\ a \ := \ \underbrace{[a; \ldots; a]}_{n \text{ veces}}$

  Tenemos que: $\texttt{make\_list}_A \ : \ \texttt{Nat} \to A \to \texttt{List}(A)$

... y tipos dependientes

- Tipo dependiente = tipo que depende de un término (de cierto tipo)
- **Ejemplo:** El tipo de las listas dependientes (= n-uplas)

Tipos $A, B \quad ::= \quad \cdots \quad | \quad \texttt{Vect}(A, N)$ (con $N : \texttt{Nat}$)

- $[] \ : \ \texttt{Vect}(A, 0)$ (para todo tipo A)
- $[\texttt{true}; \ \texttt{true}; \ \texttt{false}; \ \texttt{true}] \ : \ \texttt{Vect}(\texttt{Bool}, 4)$
- $[18; \ 42; \ 7; \ 28; \ 0; \ 13] \ : \ \texttt{Vect}(\texttt{Nat}, 6)$
- $[\texttt{succ}; \ (\lambda x : \texttt{Nat} \,.\, 0); \ \texttt{plus} \ 42] \ : \ \texttt{Vect}(\texttt{Nat} \to \texttt{Nat}, 3)$
- Sea la función $\texttt{make\_vect}_A \ n \ a \ := \ \underbrace{[a; \ldots; a]}_{n \text{ veces}}$

  Tenemos que: $\texttt{make\_vect}_A \ : \ \Pi n : \texttt{Nat} \,.\, A \to \texttt{Vect}(A, n)$

El producto dependiente

El producto dependiente

Tipos $A, B \quad ::= \quad \cdots \quad | \quad \Pi x : A \,.\, B$ (liga la variable x en B)

- $FV(\Pi x : A \,.\, B) = FV(A) \cup (FV(B) \setminus \{x\})$
- $\Pi x : A \,.\, B_x \quad =$ tipo de las funciones f que asocian a cada objeto $a : A$ un objeto $f\, a : B_a$

  $=$ producto cartesiano generalizado $\prod_{x \in A} B_x$
- Tipo flecha $A \to B =$ caso particular del producto dependiente:

  $$A \to B \quad \equiv \quad \Pi x : A \,.\, B \qquad \text{cuando } x \notin FV(B)$$
- **Ejemplo:** $\texttt{make\_vect}_A \;:\; \Pi n : \texttt{Nat} \,.\, A \to \texttt{Vect}(A, n)$

  $\equiv$

  $:\; \Pi n : \texttt{Nat} \,.\, \Pi x : A \,.\, \texttt{Vect}(A, n)$

Introducción y eliminación del producto dependiente

- Tipado de la abstracción:

$$\frac{\Gamma,\ x : A \ \vdash\ M \ :\ B}{\Gamma \ \vdash\ \lambda x : A \,.\, M \ :\ \Pi x : A \,.\, B}$$

- Tipado de la aplicación:

$$\frac{\Gamma \ \vdash\ M \ :\ \Pi x : A \,.\, B \qquad \Gamma \ \vdash\ N \ :\ A}{\Gamma \ \vdash\ M\,N \ :\ B[x := N]}$$

- **Recordatorio:** $A \to B \ \equiv\ \Pi x : A \,.\, B$ cuando $x \notin FV(B)$

Correspondencia de Curry-Howard

$$\forall x : A \,.\, B(x) \ \equiv\ \Pi x : A \,.\, B(x)$$

$$A \Rightarrow B \ \equiv\ A \to B \qquad \text{(caso no dependiente)}$$

Ejemplo

- Concatenación de listas (no dependientes):

$$\texttt{concat}_A \;:\; \texttt{List}(A) \to \texttt{List}(A) \to \texttt{List}(A)$$

- Concatenación de vectores (= listas dependientes):

$$\texttt{vconcat}_A \;:\; \Pi n : \texttt{Nat} \,.\, \texttt{Vect}(A, n) \to \Pi m : \texttt{Nat} \,.\, \texttt{Vect}(A, m) \to \texttt{Vect}(A, n + m)$$

(función con 4 argumentos)

- Dados vectores $v : \texttt{Vect}(A, 3)$ y $w : \texttt{Vect}(A, 4)$, tenemos que:

$$\begin{array}{lcl}
\texttt{vconcat}_A\ 3 & : & \texttt{Vect}(A, 3) \to \Pi m : \texttt{Nat} \,.\, \texttt{Vect}(A, m) \to \texttt{Vect}(A, 3 + m) \\
\texttt{vconcat}_A\ 3\ v & : & \Pi m : \texttt{Nat} \,.\, \texttt{Vect}(A, m) \to \texttt{Vect}(A, 3 + m) \\
\texttt{vconcat}_A\ 3\ v\ 4 & : & \texttt{Vect}(A, 4) \to \texttt{Vect}(A, 3 + 4) \\
\texttt{vconcat}_A\ 3\ v\ 4\ w & : & \texttt{Vect}(A, 3 + 4) \quad \cong \quad \texttt{Vect}(A, 7)
\end{array}$$

- Necesidad de una regla de conversión:

$$\frac{\Gamma \vdash M : A}{\Gamma \vdash M : A'} \text{ si } A \cong A'$$

Los universos (Set y Type)

¿ Cómo impedir los tipos mal formados, por ej. $\texttt{Vect}(A, 3 + \texttt{true})$?

Solución: tipar los tipos $\Rightarrow$ el universo Set

- $\texttt{Nat} \;:\; \text{Set}$
- $\texttt{Nat} \to \texttt{Nat} \;:\; \text{Set}$
- $\texttt{Vect Nat } 7 \;:\; \text{Set} \qquad (= \texttt{Vect}(\texttt{Nat}, 7))$
- $\texttt{Vect} \;:\; \text{Set} \to \texttt{Nat} \to \text{Set}$
- $\text{Set} \to \texttt{Nat} \to \text{Set} \;:\; \text{Type}$ (para evitar las paradojas)

Dos especies de tipos:

- Los tipos pequeños, de tipo Set:

  $\texttt{Nat}, \qquad \texttt{Nat} \to \texttt{Nat}, \qquad \texttt{Vect } 4, \qquad \Pi x : \texttt{Nat} \,.\; \texttt{Vect}\, x$

- Los tipos grandes ($=$ géneros), de tipo Type:

  $\text{Set}, \qquad \texttt{Nat} \to \text{Set}, \qquad \Pi x : \texttt{Nat} \,.\; \texttt{Vect}(x + 3) \to \text{Set}$

Plan

Plan

El marco lógico y sus extensiones

- Múltiples versiones de la teoría de tipos de Martin-Löf (MLTT). Todas están basadas en un marco lógico (*logical framework*), que define el cálculo lambda subyacente y sus reglas de tipado
- El marco lógico cumple las propiedades fundamentales: confluencia + β-*subject reduction* + normalización fuerte
- Luego se desarrolla el formalismo (la «teoría de tipos») añadiendo constantes con sus propias reglas de reducción y de tipado

  ¡Cuidado! Se necesita verificar que cada extensión del formalismo mantiene las 3 propiedades fundamentales: confluencia + *subject reduction* + norm. fuerte
- Aquí consideramos una presentación en el estilo de los sistemas de tipos puros (PTS), con una única categoría sintáctica de términos
- En este marco, un tipo es un término de tipo Set o Type:

  término : tipo : universo (= Set, Type)

Sintaxis de MLTT

Definición (Términos y tipos)

$$
\begin{array}{lcll}
M, N, A, B & ::= & x \mid \lambda x : A \,.\, M \mid M\,N & \text{(cálculo } \lambda \text{ a la Church)} \\
& \mid & \mathsf{Set} \mid \mathsf{Type} & \text{(universo, o suerte)} \\
& \mid & \Pi x : A \,.\, B & \text{(producto dependiente)} \\
& \mid & c & \text{(constantes)}
\end{array}
$$

- **Notaciones:** $FV(M)$ (variables libres), $M[x := N]$ (sustitución)

  En particular:
  $$FV(\lambda x : A \,.\, M) = FV(A) \cup (FV(M) \setminus \{x\})$$
  $$FV(\Pi x : A \,.\, B) = FV(A) \cup (FV(B) \setminus \{x\})$$

- Como siempre, se trabaja a menos de α-conversión

- **Ejercicio:** (1) Definir la operación de sustitución $M[x := N]$
  (2) Enunciar y demostrar el correspondiente lema de sustitución

- **Recordatorio:** $A \to B \;:\equiv\; \Pi x : A \,.\, B$ cuando $x \notin FV(B)$

Reducción

- El marco lógico sólo viene con la regla de β-reducción:

$$(\beta) \qquad (\lambda x : A \,.\, M)\, N \;\succ\; M[x := N]$$

  + clausura contextual

- Luego se extiende el sistema con constantes (notación: c, d, etc.) acompañadas con su tipo y sus reglas de reducción: las δ-reglas
- Las δ-reglas son en general de la forma:

$$\begin{array}{lcl} d \;\cdots\; (c_1 \;\cdots) \;\cdots & \succ & \cdots \\ \vdots & & \vdots \\ d \;\cdots\; (c_n \;\cdots) \;\cdots & \succ & \cdots \end{array}$$

  («definición por casos») donde:
  - d es un destructor que opera sobre cierto tipo inductivo
  - $c_1, \ldots, c_n$ son los constructores de dicho tipo inductivo

- En lo siguiente, sólo consideraremos reglas que mantienen las 3 propiedades fundamentales: confluencia + S.R. + norm. fuerte

Tipado (1/2)

- Sistema de tipado parametrizado por:
  - Una función $c \mapsto \mathrm{ty}(c)$ que asocia a cada constante c su tipo $\mathrm{ty}(c)$ (a priori, $\mathrm{ty}(c)$ es un término cerrado cualquiera)
  - La relación de conversión $M \cong M'$ inducida por la regla de β-reducción y las δ-reglas asociadas a las constantes del sistema
- Contexto de tipado = lista ordenada de declaraciones de la forma

$$\Gamma \quad \equiv \quad x_1 : A_1, \ldots, x_n : A_n$$

donde $\begin{cases} x_1, \ldots, x_n \text{ son variables distintas a pares} \\ A_1, \ldots, A_n \text{ son términos cualesquiera} \end{cases}$

- Sistema de tipado basado en dos juicios:
  - Un juicio de buena formación de contexto

    $\vdash \Gamma$ context «El contexto Γ está bien formado»
  - Un juicio de tipado

    $\Gamma \vdash M : A$ «En el contexto Γ, el término M tiene tipo A»

Tipado (2/2)

- Reglas del juicio $\vdash \Gamma$ context (buena formación de contexto)

$$\frac{}{\vdash \emptyset \text{ context}} \qquad \frac{\Gamma \vdash A : s}{\vdash \Gamma, x : A \text{ context}} \text{ si } x \notin \text{dom}(\Gamma),\ s \in \{\text{Set}, \text{Type}\}$$

- Reglas del juicio $\Gamma \vdash M : A$ (tipado)

$$\frac{\vdash \Gamma \text{ context}}{\Gamma \vdash x : A} \text{ si } (x:A) \in \Gamma \qquad \frac{\vdash \Gamma \text{ context}}{\Gamma \vdash c : \text{ty}(c)} \qquad \frac{\vdash \Gamma \text{ context}}{\Gamma \vdash \text{Set} : \text{Type}}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x : A \,.\, M : \Pi x : A \,.\, B} \qquad \frac{\Gamma \vdash M : \Pi x : A \,.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\, N : B[x := N]}$$

$$\frac{\Gamma \vdash A : s_1 \qquad \Gamma, x : A \vdash B : s_2}{\Gamma \vdash \Pi x : A \,.\, B : \max(s_1, s_2)} \text{ si } s_1, s_2 \in \{\text{Set}, \text{Type}\}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A' : s}{\Gamma \vdash M : A'} \text{ si } A \cong A',\ s \in \{\text{Set}, \text{Type}\}$$

(Con la convención Set $<$ Type)

Ejemplo de derivación

Dadas constantes $\texttt{Nat} : \text{Set}$, $\texttt{0} : \texttt{Nat}$, $\texttt{S} : \texttt{Nat} \to \texttt{Nat}$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash \emptyset \text{ context}}}{\emptyset \vdash \text{Set} : \text{Type}}}{\vdash X : \text{Set context}}}{X : \text{Set} \vdash X : \text{Set}}}{\vdash X : \text{Set},\ x : X \text{ context}}}{X : \text{Set},\ x : X \vdash x : X}}{X : \text{Set} \vdash \lambda x : X \,.\, x : X \to X}}{\emptyset \vdash \lambda X : \text{Set} \,.\, \lambda x : X \,.\, x : \Pi x : \text{Set} \,.\, X \to X} \qquad \dfrac{\dfrac{\dfrac{}{\vdash \emptyset \text{ context}}}{\emptyset \vdash \texttt{Nat} : \text{Set}} \qquad \dfrac{\dfrac{\dfrac{}{\vdash \emptyset \text{ context}}}{\emptyset \vdash \texttt{Nat} : \text{Set}}}{\vdash x : \texttt{Nat} \text{ context}}}{x : \texttt{Nat} \vdash \texttt{Nat} : \text{Set}}}{\emptyset \vdash \texttt{Nat} \to \texttt{Nat} : \text{Set}}$$

$$\dfrac{\emptyset \vdash \lambda X : \text{Set} \,.\, \lambda x : X \,.\, x : \Pi x : \text{Set} \,.\, X \to X \qquad \emptyset \vdash \texttt{Nat} \to \texttt{Nat} : \text{Set}}{\emptyset \vdash (\lambda X : \text{Set} \,.\, \lambda x : X \,.\, x)\,(\texttt{Nat} \to \texttt{Nat}) : (\texttt{Nat} \to \texttt{Nat}) \to \texttt{Nat} \to \texttt{Nat}} \qquad \dfrac{\dfrac{}{\vdash \emptyset \text{ context}}}{\emptyset \vdash \texttt{S} : \texttt{Nat} \to \texttt{Nat}}$$

$$\dfrac{\emptyset \vdash (\lambda X : \text{Set} \,.\, \lambda x : X \,.\, x)\,(\texttt{Nat} \to \texttt{Nat}) : (\texttt{Nat} \to \texttt{Nat}) \to \texttt{Nat} \to \texttt{Nat} \qquad \emptyset \vdash \texttt{S} : \texttt{Nat} \to \texttt{Nat}}{\emptyset \vdash (\lambda X : \text{Set} \,.\, \lambda x : X \,.\, x)\,(\texttt{Nat} \to \texttt{Nat})\,\texttt{S} : \texttt{Nat} \to \texttt{Nat}}$$

- Las derivaciones de la teoría de tipos son en general muy largas, debido a la necesidad de justificar la buena formación del contexto para cada variable x, cada constante c, y cada ocurrencia del universo Set
- Por suerte, cuando el sistema es confluente y normalizante, la relación $\Gamma \vdash M : A$ es decidible: no se necesita mantener las derivaciones

Ejemplos: identidad monomórfica y polimórfica

Dadas constantes $\mathtt{Nat} : \mathsf{Set}$, $\mathtt{0} : \mathtt{Nat}$, $\mathtt{S} : \mathtt{Nat} \to \mathtt{Nat}$, tenemos que:

- $\vdash \lambda x : \mathtt{Nat} \,.\, x \;:\; \mathtt{Nat} \to \mathtt{Nat}$
- $\vdash \mathtt{Nat} \to \mathtt{Nat} \;:\; \mathsf{Set}$ (tipo pequeño)

- $\vdash \lambda x : \mathtt{Nat} \to \mathtt{Nat} \,.\, x \;:\; (\mathtt{Nat} \to \mathtt{Nat}) \to \mathtt{Nat} \to \mathtt{Nat}$
- $\vdash (\mathtt{Nat} \to \mathtt{Nat}) \to \mathtt{Nat} \to \mathtt{Nat} \;:\; \mathsf{Set}$ (tipo pequeño)

- $\vdash \lambda X : \mathsf{Set} \,.\, \lambda x : X \,.\, x \;:\; \Pi X : \mathsf{Set} \,.\, X \to X$
- $\vdash \Pi X : \mathsf{Set} \,.\, X \to X \;:\; \mathsf{Type}$ (tipo grande)

- $\vdash (\lambda X : \mathsf{Set} \,.\, \lambda x : X \,.\, x)\, \mathtt{Nat} \;:\; \mathtt{Nat} \to \mathtt{Nat}$
- $\vdash (\lambda X : \mathsf{Set} \,.\, \lambda x : X \,.\, x)\, (\mathtt{Nat} \to \mathtt{Nat}) \;:\; (\mathtt{Nat} \to \mathtt{Nat}) \to \mathtt{Nat} \to \mathtt{Nat}$

- Pero $(\lambda X : \mathsf{Set} \,.\, \lambda x : X \,.\, x)\, (\Pi X : \mathsf{Set} \,.\, X \to X)$ está mal tipado

  $\Rightarrow$ En la teoría de tipos de Martin-Löf, el polimorfismo es predicativo

- $\vdash \lambda f : (\Pi X : \mathsf{Set} \,.\, X \to X) \,.\, f \;:\; (\Pi X : \mathsf{Set} \,.\, X \to X) \to (\Pi X : \mathsf{Set} \,.\, X \to X)$

Ejemplos: calculando con los tipos

En la teoría de tipos de Martin-Löf, los tipos son términos particulares
$\Rightarrow$ también se puede calcular con los tipos:

- Sea $\mathtt{id}_{\mathsf{Set}} :\equiv \lambda X : \mathsf{Set}\,.\,X \;:\; \mathsf{Set} \to \mathsf{Set}$

  Tenemos que $\mathtt{id}_{\mathsf{Set}}\ \mathtt{Nat}\ (:\mathsf{Set}) \succ \mathtt{Nat}$
  (suponiendo $\mathtt{Nat} : \mathsf{Set}$)

- Sea $\mathtt{arrow} :\equiv \lambda X, Y : \mathsf{Set}\,.\,X \to Y \;:\; \mathsf{Set} \to \mathsf{Set} \to \mathsf{Set}$

  Tenemos que $\mathtt{arrow}\ \mathtt{Nat}\ \mathtt{Bool}\ (:\mathsf{Set}) \succ^2 \mathtt{Nat} \to \mathtt{Bool}$
  (suponiendo $\mathtt{Nat}, \mathtt{Bool} : \mathsf{Set}$)

- Más generalmente, sea: $\mathtt{marrow} \;:\; \mathtt{Nat} \to \mathsf{Set} \to \mathsf{Set} \to \mathsf{Set}$

  con las δ-reglas:
  $$\begin{aligned} \mathtt{marrow}\ \mathtt{0}\ A\ B &\succ B \\ \mathtt{marrow}\ (\mathtt{S}\ n)\ A\ B &\succ A \to \mathtt{marrow}\ n\ A\ B \end{aligned}$$

  Para todos $A, B : \mathsf{Set}$ y $n \in \mathbb{N}$, tenemos que:

  $$\mathtt{marrow}\ (\mathtt{S}^n\ \mathtt{0})\ A\ B \;\succ^*\; \underbrace{A \to \cdots \to A}_{n} \to B$$

Ejemplo con vectores (= listas dependientes)

- Sean las constantes

$$
\begin{array}{lcl}
\texttt{Nat} & : & \texttt{Set} \\
\texttt{0} & : & \texttt{Nat} \\
\texttt{S} & : & \texttt{Nat} \to \texttt{Nat} \qquad \text{(Notación: } 1 :\equiv \texttt{S}\,\texttt{0},\quad 2 :\equiv \texttt{S}\,1,\quad \text{etc.)} \\
\texttt{plus} & : & \texttt{Nat} \to \texttt{Nat} \to \texttt{Nat} \\
 & & \\
\texttt{Vect} & : & \texttt{Set} \to \texttt{Nat} \to \texttt{Set} \\
\texttt{vnil} & : & \Pi A : \texttt{Set} \,.\, \texttt{Vect}\ A\ \texttt{0} \\
\texttt{vcons} & : & \Pi A : \texttt{Set} \,.\, \Pi n : \texttt{Nat} \,.\, A \to \texttt{Vect}\ A\ n \to \texttt{Vect}\ A\ (\texttt{S}\ n) \\
 & & \\
\texttt{vconcat} & : & \Pi A : \texttt{Set} \,.\, \Pi n : \texttt{Nat} \,.\, \texttt{Vect}\ A\ n \to \\
 & & \qquad \Pi m : \texttt{Nat} \,.\, \texttt{Vect}\ A\ m \to \texttt{Vect}\ A\ (\texttt{plus}\ n\ m)
\end{array}
$$

  con las δ-reglas:

$$
\begin{array}{rcl}
\texttt{plus}\ \texttt{0}\ m & \succ & m \\
\texttt{plus}\ (\texttt{S}\ n)\ m & \succ & \texttt{S}\ (\texttt{plus}\ n\ m) \\
 & & \\
\texttt{vconcat}\ A\ \texttt{0}\ (\texttt{vnil}\ A)\ m\ v & \succ & v \\
\texttt{vconcat}\ A\ (\texttt{S}\ n)\ (\texttt{vcons}\ A\ n\ u)\ m\ v & \succ & \texttt{vcons}\ A\ (\texttt{plus}\ n\ m)\ (\texttt{vconcat}\ n\ u\ m\ v)
\end{array}
$$

- Dados $A : \text{Set}$, $u : \texttt{Vect}\ A\ 3$, $v : \texttt{Vect}\ A\ 4$, tenemos que:

$$\texttt{vconcat}\ A\ 3\ u\ 4\ v \quad : \quad \texttt{Vect}\ A\ 7 \qquad \text{(por conversión)}$$

Propiedades básicas (1/3)

Notación: $\Gamma' \sqsubseteq \Gamma \quad \equiv \quad \Gamma'$ es un prefijo de Γ

Recordatorio: Las declaraciones de un contexto Γ son ordenadas

Lema (Buena formación)

1. Si $\Gamma \vdash M : A$, entonces $\vdash \Gamma$ context
2. Si $\vdash \Gamma$ context, entonces $\vdash \Gamma'$ context para todo $\Gamma' \sqsubseteq \Gamma$

Demostración. Más generalmente, se demuestra por inducción sobre el tamaño de las derivaciones involucradas que:

1. Toda derivación de $\Gamma \vdash M : A$ contiene una subderivación de $\vdash \Gamma$ context
2. Toda derivación de $\vdash \Gamma$ context contiene una subderivación de $\vdash \Gamma'$ context para cada prefijo $\Gamma' \sqsubseteq \Gamma$ □

Dicho de otro modo:

1. El contexto de un juicio de tipado derivable siempre está bien formado
2. Todo prefijo de un contexto bien formado también está bien formado

Propiedades básicas (2/3)

Lema (Variables libres)

1. Si $\vdash x_1 : A_1, \ldots, x_n : A_n$ context, entonces
   - $FV(A_i) \subseteq \{x_1, \ldots, x_{i-1}\}$ para todo $i \in [1..n]$
2. Si $x_1 : A_1, \ldots, x_n : A_n \vdash M : A$, entonces
   - $FV(A_i) \subseteq \{x_1, \ldots, x_{i-1}\}$ para todo $i \in [1..n]$
   - $FV(M) \subseteq \{x_1, \ldots, x_n\}$ y $FV(A) \subseteq \{x_1, \ldots, x_n\}$

Demostración. Por inducción mutua sobre las derivaciones involucradas. □

Dicho de otro modo, los juicios de tipado derivables son de la forma:

$$
\begin{array}{l}
x_1 : A_1, \\
x_2 : A_2(x_1), \\
x_3 : A_3(x_1, x_2), \\
\quad \vdots \\
x_n : A_n(x_1, \ldots, x_{n-1}) \\
\vdash \; M(x_1, \ldots, x_n) \; : \; A(x_1, \ldots, x_n)
\end{array}
$$

Propiedades básicas (3/3)

Recordatorio: Dados contextos Γ, Γ', se escribe $\Gamma \subseteq \Gamma'$ cuando $(x : A) \in \Gamma$ implica $(x : A) \in \Gamma'$ para toda declaración $(x : A)$

Lema (Debilitamiento)

Si $\Gamma \vdash M : A$, $\Gamma \subseteq \Gamma'$ y $\vdash \Gamma'$ context, entonces $\Gamma' \vdash M : A$

Demostración. Por inducción sobre la derivación de $\Gamma \vdash M : A$. □

Lema (Sustitutividad)

1. Si $\vdash \Gamma, x : A, \Gamma'$ context y $\Gamma \vdash N : A$, entonces $\vdash \Gamma, \Gamma'[x := N]$ context
2. Si $\Gamma, x : A, \Gamma' \vdash M : B$ y $\Gamma \vdash N : A$, entonces $\Gamma, \Gamma'[x := N] \vdash M[x := N] : B[x := N]$

Demostración. Por inducción mutua sobre las derivaciones de los juicios $\vdash \Gamma, x : A, \Gamma'$ context y $\Gamma, x : A, \Gamma' \vdash M : B$, usando la propiedad de debilitamiento para tratar el caso donde $M \equiv x$ □

El lema de inversión y sus consecuencias (1/2)

Lema de inversión

- $\Gamma \vdash x : C \quad \Rightarrow \quad \exists A \begin{cases} (x : A) \in \Gamma \\ C \cong A \end{cases}$
- $\Gamma \vdash c : C \quad \Rightarrow \quad C \cong \mathsf{ty}(c)$
- $\Gamma \vdash \mathsf{Set} : C \quad \Rightarrow \quad C \equiv \mathsf{Type}$
- $\Gamma \vdash \lambda x : A \,.\, M : C \quad \Rightarrow \quad \exists B \begin{cases} \Gamma, x : A \vdash M : B \\ C \cong \Pi x : A \,.\, B \end{cases}$
  (con $x \notin \mathrm{dom}(\Gamma)$)
- $\Gamma \vdash M\, N : C \quad \Rightarrow \quad \exists A, B \begin{cases} \Gamma \vdash M : \Pi x : A \,.\, B \\ \Gamma \vdash N : A \\ C \cong B[x := N] \end{cases}$
- $\Gamma \vdash \Pi x : A \,.\, B : C \quad \Rightarrow \quad \exists s_1, s_2 \begin{cases} \Gamma \vdash A : s_1 \\ \Gamma, x : A \vdash B : s_2 \\ C \cong \max(s_1, s_2) \end{cases}$
  (con $x \notin \mathrm{dom}(\Gamma)$)

Demostración. Por inducción sobre la derivación. □

El lema de inversión y sus consecuencias (2/2)

- Si la reducción ($\beta + \delta$-reglas) es confluente, entonces:

Corolario (Unicidad del tipo)

Si $\Gamma \vdash M : A$ y $\Gamma \vdash M : A'$, entonces $A \cong A'$

Demostración. Por inducción sobre M, usando el lema de inversión en cada etapa, así como la confluencia de la reducción en el caso donde M es una aplicación. □

Ejercicio: ¿Por qué se necesita la confluencia?

Corolario (β-*subject reduction*)

Si $\Gamma \vdash M : A$ y $M \succ_\beta M'$, entonces $\Gamma \vdash M' : A$

Demostración: Ejercicio.

- **Obs.:** Para demostrar la propiedad de *subject reduction* completa, se hace un razonamiento análogo para cada δ-regla $M \succ M'$ (usando de vuelta el lema de inversión)

Tipos de los tipos

- Hasta ahora, no supusimos nada sobre la función $c \mapsto \mathsf{ty}(c)$ que parametriza el sistema de tipado
- A partir de ahora, se supone que para cada constante c, tenemos que $\emptyset \vdash \mathsf{ty}(c) : s$ para algún $s \in \{\mathsf{Set}, \mathsf{Type}\}$

  (Esto implica en particular que $\mathsf{ty}(c)$ es un tipo cerrado)

Lema (Tipo de los tipos)

Bajo la hipótesis anterior, si $\Gamma \vdash M : A$, entonces:

- o bien $A \equiv \mathsf{Type}$
- o bien $\Gamma \vdash A : s$ para algún $s \in \{\mathsf{Set}, \mathsf{Type}\}$

Demostración. Por inducción sobre la derivación de $\Gamma \vdash M : A$, usando el lema de inversión en el caso de la regla de tipado la aplicación. □

Ejercicio: Detallar todos los casos, y en particular el caso de la aplicación

Verificación e inferencia de tipo (1/4)

Se consideran los siguientes tres problemas:

1. **Verificación de contexto:**
   Dado Γ, determinar si el juicio $\vdash \Gamma$ context es derivable o no
2. **Verificación de tipo:**
   Dados Γ, M, A, determinar si el juicio $\Gamma \vdash M : A$ es derivable o no
3. **Inferencia de tipo:**
   Dados Γ y M, determinar si existe A tal que $\Gamma \vdash M : A$
   (y devolver tal tipo A cuando existe)

Teorema

Si la $\beta\delta$-reducción es confluente, y si el sistema es fuertemente normalizante, entonces los tres problemas anteriores son decidibles

- La complejidad teórica de los correspondientes algoritmos es la del test de conversión $A \cong A'$ (decidible si confluencia + norm. fuerte)

Verificación e inferencia de tipo (2/4)

A partir de:

- la función de normalización $M \mapsto \downarrow M$
- el test de conversión $M_1 \cong M_2 \quad := \quad \downarrow M_1 \equiv \downarrow M_2$

Se implementan 4 funciones (mutuamente recursivas):

- **InferType**$(\Gamma,\ M)$: calcular A (si existe) tal que $\Gamma \vdash M : A$, bajo la hipótesis que $\vdash \Gamma$ context
- **InferSort**$(\Gamma,\ A)$: calcular $s \in \{\text{Set}, \text{Type}\}$ (si existe) tal que $\Gamma \vdash A : s$, bajo la hipótesis que $\vdash \Gamma$ context
- **CheckType**$(\Gamma,\ M,\ A)$: verificar si $\Gamma \vdash M : A$, bajo la hipótesis que $\vdash \Gamma$ context
- **CheckCtx**(Γ) : verificar si $\vdash \Gamma$ context

Obs.: Por Curry-Howard, **CheckType** también ¡verifica las pruebas!

Verificación e inferencia de tipo (3/4)

InferType$(\Gamma,\ M)\ :=$

En función de la forma del término M:

- $M \equiv x$**:** Si $(x : A) \in \Gamma$ para algún A: devolver A
  si no: devolver "no tipable"
- $M \equiv c$**:** Devolver ty(c)
- $M \equiv$ Set**:** Devolver Type
- $M \equiv$ Type**:** Devolver "no tipable"
- $M \equiv \lambda x : A\,.\,M_1$**:** Sea $s := $ **InferSort**$(\Gamma,\ A)$ (cuando existe)
  Sea $B := $ **InferType**$((\Gamma, x : A),\ M_1)$ (cuando existe)
  Si s, B existen: devolver $\Pi x : A\,.\,B$;
  Si no: devolver "no tipable"
- $M \equiv M_1\ M_2$**:** Sea $A_1 := $ **InferType**$(\Gamma,\ M_1)$ (cuando existe)
  Sea $A_2 := $ **InferType**$(\Gamma,\ M_2)$ (cuando existe)
  Si A_1, A_2 existen y $A_1 \cong \Pi x : A_2\,.\,B$
  para algún B: devolver $B[x := M_2]$;
  Si no: devolver "no tipable"
- $M \equiv \Pi x : A\,.\,B$**:** Sea $s_1 := $ **InferSort**$(\Gamma,\ A)$ (cuando existe)
  Sea $s_2 := $ **InferSort**$((\Gamma, x : A),\ B)$ (cuando existe)
  Si s_1, s_2 existen: devolver $\max(s_1, s_2)$
  Si no: devolver "no tipable"

Verificación e inferencia de tipo (4/4)

InferSort$(\Gamma,\ A)\ :=$

Sea $s\ :=\ \downarrow$**InferType**$(\Gamma,\ A)$ (cuando existe)
Si s existe y $s \in \{\mathsf{Set}, \mathsf{Type}\}$: devolver s
Si no: devolver "no es un tipo"

CheckType$(\Gamma,\ M,\ A)\ :=$

Sea $A'\ :=$ **InferType**$(\Gamma,\ M)$ (cuando existe)
Sea $s\ :=$ **InferSort**$(\Gamma,\ A)$ (cuando existe)
Si (A' existe y $A' \equiv A \equiv \mathsf{Type}$) o
(A', s existen y $A' \cong A$): devolver "derivable";
Si no: devolver "no derivable"

CheckCtx$(\Gamma)\ :=$

En función de la forma del contexto Γ:

- $\Gamma \equiv \emptyset$: devolver "bien formado"
- $\Gamma \equiv \Gamma_0, x : A$: Si **CheckCtx**$(\Gamma_0)$ y $x \notin \mathrm{dom}(\Gamma_0)$:
  Sea $s\ :=$ **InferSort**$(\Gamma_0,\ A)$ (cuando existe)
  Si s existe: devolver "bien formado"
  Si no: devolver "mal formado"
  Si no: devolver "mal formado"

Plan

Añadiendo constantes

- Tres especies de constantes:
  - Constantes de (familias de) tipos (`Nat`, `List`, `Vect`)
  - Constructores (`0`, `S`, `nil`, `cons`)
  - Destructores (`if`, `nat_rec`, `list_rec`)
- Cada destructor d viene con δ-reglas de la forma

$$\begin{array}{llll} d & \cdots & (c_1 & \cdots) & \cdots & \succ & \cdots \\ d & \cdots & (c_2 & \cdots) & \cdots & \succ & \cdots \\ & & \vdots & & & & \vdots \\ d & \cdots & (c_n & \cdots) & \cdots & \succ & \cdots \end{array}$$

  donde $c_1, \ldots, c_n$ son los constructores del tipo inductivo destruido por d

- **Requísito:** Mantener los tres invariantes:

  Confluencia + *$\beta\delta$-Subject reduction* + Norm. fuerte

El tipo de los enteros naturales

$$\begin{array}{lcl} \texttt{Nat} & : & \texttt{Set} \\ \texttt{0} & : & \texttt{Nat} \\ \texttt{S} & : & \texttt{Nat} \to \texttt{Nat} \\ \texttt{nat\_elim} & : & \Pi X : \texttt{Set} \,.\, X \to (\texttt{Nat} \to X \to X) \to \texttt{Nat} \to X \end{array}$$

(`nat_elim` = recursor del sistema T)

δ-reglas asociadas

$$\begin{array}{lcl} \texttt{nat\_elim}\, X\, x_0\, f\, \texttt{0} & \succ & x_0 \\ \texttt{nat\_elim}\, X\, x_0\, f\, (\texttt{S}\, n) & \succ & f\, n\, (\texttt{nat\_elim}\, X\, x_0\, f\, n) \end{array}$$

- `nat_elim` permite implementar las funciones usuales:

$$\begin{array}{lcl} \texttt{plus} & :\equiv & \lambda n, m : \texttt{Nat} \,.\, \texttt{nat\_elim Nat}\ m\ (\lambda \_, z : \texttt{Nat} \,.\, \texttt{S}\ z)\ n \\ \texttt{mult} & :\equiv & \lambda n, m \,.\, \texttt{Nat nat\_elim Nat 0}\ (\lambda \_, z : \texttt{Nat} \,.\, z + m)\ n \\ \texttt{pred} & :\equiv & \texttt{nat\_elim Nat 0}\ (\lambda z, \_ : \texttt{Nat} \,.\, z) \end{array}$$

- **Pregunta:** ¿Cómo demostrar las propiedades de estas funciones?

El esquema de eliminación dependiente

$$\texttt{nat\_elim} \;:\; \Pi X : \mathsf{Set} \;.\; X \to (\texttt{Nat} \to X \to X) \to \texttt{Nat} \to X$$

$$\underset{x_0}{X\,0} \xrightarrow[f\,0]{} X\,1 \xrightarrow[f\,1]{} X\,2 \xrightarrow[f\,2]{} \cdots \xrightarrow[f\,(n-1)]{} \underset{x_n}{X\,n}$$

$$\begin{array}{lcl} X & : & \texttt{Nat} \to \mathsf{Set} \\ x_0 & : & X\,0 \\ f & : & \Pi p : \texttt{Nat} \,.\; X\,p \to X\,(\mathtt{S}\,p) \\ x_n & : & X\,n \end{array}$$

Recursor dependiente

$$\texttt{nat\_rec} \;:\; \Pi X : \texttt{Nat} \to \mathsf{Set} \,. \quad X\;\mathtt{0} \;\to\; (\Pi p : \texttt{Nat} \,.\; X\,p \to X\,(\mathtt{S}\,p)) \;\to\; \Pi n : \texttt{Nat} \,.\; X\,n$$

$=$ Principio de inducción (Curry-Howard)

- `nat_rec` tiene las mismas δ-reglas que `nat_elim`
- `nat_elim` se puede implementar a partir de `nat_rec`

Los enteros naturales, de vuelta

$$
\begin{array}{lcl}
\mathtt{Nat} & : & \mathsf{Set} \\
\mathtt{0} & : & \mathtt{Nat} \\
\mathtt{S} & : & \mathtt{Nat} \to \mathtt{Nat} \\
\mathtt{nat\_rec} & : & \Pi X : \mathtt{Nat} \to \mathsf{Set}\,. \\
 & & \quad X\,\mathtt{0} \to (\Pi p : \mathtt{Nat}\,.\, X\,p \to X(\mathtt{S}\,p)) \to \Pi n : \mathtt{Nat}\,.\, X\,n
\end{array}
$$

δ-reglas asociadas

$$
\begin{array}{lcl}
\mathtt{nat\_rec}\,X\,x_0\,f\,\mathtt{0} & \succ & x_0 \\
\mathtt{nat\_rec}\,X\,x_0\,f\,(\mathtt{S}\,n) & \succ & f\,n\,(\mathtt{nat\_rec}\,X\,x_0\,f\,n)
\end{array}
$$

- Versión no dependiente de `nat_rec`:

$$
\begin{array}{lcl}
\mathtt{nat\_elim} & : & \Pi X : \mathsf{Set}\,.\, X \to (\mathtt{Nat} \to X \to X) \to \mathtt{Nat} \to X \\
 & :\equiv & \lambda X : \mathsf{Set}\,.\, \mathtt{nat\_rec}\,(\lambda \_ : \mathtt{Nat}\,.\, X)
\end{array}
$$

El tipo de los booleanos

$$
\begin{array}{lcl}
\texttt{Bool} & : & \mathsf{Set} \\
\texttt{true} & : & \texttt{Bool} \\
\texttt{false} & : & \texttt{Bool} \\
\texttt{bool\_rec} & : & \Pi X : \texttt{Bool} \to \mathsf{Set}\,. \\
 & & \quad X\,\texttt{true} \to X\,\texttt{false} \to \Pi b : \texttt{Bool}\,.\, X\,b
\end{array}
$$

δ-reglas asociadas

$$
\begin{array}{lcl}
\texttt{bool\_rec}\,X\,x\,y\,\texttt{true} & \succ & x \\
\texttt{bool\_rec}\,X\,x\,y\,\texttt{false} & \succ & y
\end{array}
$$

- Versión no dependiente:

$$
\begin{array}{lcl}
\texttt{if} & : & \Pi X : \mathsf{Set}\,.\,\texttt{Bool} \to X \to X \to X \\
 & :\equiv & \lambda X : \mathsf{Set}\,.\,\texttt{bool\_rec}\,(\lambda \_ : \texttt{Bool}\,.\,X)
\end{array}
$$

El producto cartesiano $A \times B \quad (= A \wedge B)$

$$\texttt{Prod} \;:\; \mathsf{Set} \to \mathsf{Set} \to \mathsf{Set} \qquad \text{(Notación: } A \times B \;:\equiv\; \texttt{Prod}\, A\, B\text{)}$$

$$\texttt{pair} \;:\; \Pi A, B : \mathsf{Set}\,.\, A \to B \to A \times B \qquad \text{(Notación: } \langle a, b\rangle \;:\equiv\; \texttt{pair}\, A\, B\, a\, b\text{)}$$

$$\texttt{prod\_rec} \;:\; \Pi A, B : \mathsf{Set}\,.\, \Pi X : A \times B \to \mathsf{Set}\,. \\ (\Pi x : A\,.\,\Pi y : B\,.\;\; X\,\langle x, y\rangle) \to \Pi p : A \times B\,.\, X\, p$$

δ-regla asociada

$$\texttt{prod\_rec}\, A\, B\, X\, f\, \langle a, b\rangle \;\;\succ\;\; f\, a\, b$$

- Eliminación no dependiente y proyecciones:

$$\begin{aligned} \texttt{prod\_elim} \;&:\; \Pi A, B, X : \mathsf{Set}\,.\, (A \to B \to X) \to A \times B \to X \\ &:\equiv\; \lambda A, B, X : \mathsf{Set}\,.\, \texttt{prod\_rec}\, A\, B\, (\lambda \_ : A \times B\,.\, X) \end{aligned}$$

$$\begin{aligned} \texttt{fst} \;&:\; \Pi A, B : \mathsf{Set}\,.\, A \times B \to A \;:\equiv\; \cdots \\ \texttt{snd} \;&:\; \Pi A, B : \mathsf{Set}\,.\, A \times B \to B \;:\equiv\; \cdots \quad \text{(Ejercicio)} \end{aligned}$$

La suma directa $A + B \quad (= A \vee B)$

$$\begin{array}{lcll}
\texttt{Sum} & : & \mathsf{Set} \to \mathsf{Set} \to \mathsf{Set} & \text{(Notación: } A + B :\equiv \texttt{Sum}\ A\ B\text{)} \\
\texttt{inl} & : & \Pi A, B : \mathsf{Set}\,.\, A \to A + B & \text{(Notación: } \texttt{inl}(a) :\equiv \texttt{inl}\ A\ B\ a\text{)} \\
\texttt{inr} & : & \Pi A, B : \mathsf{Set}\,.\, B \to A + B & \text{(Notación: } \texttt{inr}(b) :\equiv \texttt{inr}\ A\ B\ b\text{)} \\
\texttt{sum_rec} & : & \Pi A, B : \mathsf{Set}\,.\, \Pi X : A + B \to \mathsf{Set}\,. & \\
 & & (\Pi x : A\,.\ X\,(\texttt{inl}(x))) \to (\Pi y : B\,.\ X\,(\texttt{inr}(y))) \to & \\
 & & \Pi s : A + B\,.\ X\ s &
\end{array}$$

δ-reglas asociadas

$$\begin{array}{rcl}
\texttt{sum_rec}\ A\ B\ X\ f\ g\,(\texttt{inl}(a)) & \succ & f\ a \\
\texttt{sum_rec}\ A\ B\ X\ f\ g\,(\texttt{inr}(b)) & \succ & g\ b
\end{array}$$

- Eliminación no dependiente:

$$\begin{array}{lcl}
\texttt{sum_elim} & : & \Pi A, B : \mathsf{Set}\,.\, \Pi X : \mathsf{Set}\,.\, (A \to X) \to (B \to X) \to A + B \to X \\
 & \equiv & \lambda A, B : \mathsf{Set}\,.\, \lambda X : \mathsf{Set}\,.\, \texttt{sum_rec}\ A\ B\ (\lambda _ : A + B\,.\, X)
\end{array}$$

El tipo vacío $\mathbf{0}$ ($= \bot$) y el tipo unitario $\mathbf{1}$ ($= \top$)

- El tipo vacío $\mathbf{0}$ ($=$ proposición absurda $\bot$)

$$\begin{array}{lcl} \mathbf{0} & : & \mathsf{Set} \\ \multicolumn{3}{l}{\text{(Ningún constructor)}} \\ \texttt{empty\_rec} & : & \Pi X : \mathbf{0} \to \mathsf{Set} \,.\, \Pi e : \mathbf{0} \,.\, X\, e \end{array}$$

(Ninguna δ-regla asociada)

- El tipo unitario $\mathbf{1}$ ($=$ proposición obvia $\top$)

$$\begin{array}{lcl} \mathbf{1} & : & \mathsf{Set} \\ \langle\rangle & : & \mathbf{1} \\ \texttt{unit\_rec} & : & \Pi X : \mathbf{1} \to \mathsf{Set} \,.\, X\, \langle\rangle \to \Pi u : \mathbf{1} \,.\, X\, u \end{array}$$

δ-regla asociada

$$\texttt{unit\_rec}\, X\, x\, \langle\rangle \quad \succ \quad x$$

La suma dependiente $\Sigma x : A \,.\, B\, x \quad (= \exists x : A \,.\, B\, x)$

$$\begin{array}{lcll}
\texttt{DSum} & : & \Pi A : \mathsf{Set} \,.\, (A \to \mathsf{Set}) \to \mathsf{Set} & (\Sigma x : A \,.\, B\, x \;:\equiv\; \texttt{DSum}\, A\, B) \\
\texttt{dpair} & : & \Pi A : \mathsf{Set} \,.\, \Pi B : A \to \mathsf{Set} \,. & \\
 & & \quad \Pi x : A \,.\, B\, x \to \Sigma x : A \,.\, B\, x & (\langle a, b\rangle \;:\equiv\; \texttt{dpair}\, A\, B\, a\, b) \\
\texttt{dsum_rec} & : & \Pi A : \mathsf{Set} \,.\, \Pi B : A \to \mathsf{Set} \,.\, \Pi X : (\Sigma x : A \,.\, B) \to \mathsf{Set} \,. & \\
 & & \quad (\Pi x : A \,.\, \Pi y : B\, x \,.\; X\, \langle x, y\rangle) \to \Pi z : (\Sigma x : A \,.\, B\, x) \,.\, X\, z &
\end{array}$$

δ-regla asociada

$$\texttt{dsum\_rec}\, A\, B\, X\, f\, \langle a, b\rangle \;\succ\; f\, a\, b$$

- Elimination no dependiente y proyecciones (Ejercicio):

$$\begin{array}{lcll}
\texttt{dsum_elim} & : & \Pi A : \mathsf{Set} \,.\, \Pi B : A \to \mathsf{Set} \,.\, \Pi X : \mathsf{Set} \,. & \\
 & & \quad (\Pi x : A \,.\, B\, x \to X) \to (\Sigma x : A \,.\, B\, x) \to X & :\equiv \quad \cdots \\
\texttt{dfst} & : & \Pi A : \mathsf{Set} \,.\, \Pi B : A \to \mathsf{Set} \,.\, (\Sigma x : A \,.\, B\, x) \to A & :\equiv \quad \cdots \\
\texttt{dsnd} & : & \Pi A : \mathsf{Set} \,.\, \Pi B : A \to \mathsf{Set} \,.\, \Pi s : (\Sigma x : A \,.\, B\, x) \,.\, B\, (\texttt{dfst}\, A\, B\, s) & :\equiv \quad \cdots
\end{array}$$

- **Obs.:** Si $x \notin FV(B)$, entonces $\Sigma x : A \,.\, B \;\sim\; A \times B$ (iso)

El tipo identidad $x =_A y$ (1/3)

$$\begin{array}{lcl}
\texttt{Eq} & : & \Pi A : \mathsf{Set} \,.\, A \to A \to \mathsf{Set} \qquad \text{(Notación: } x =_A y \;:\equiv\; \texttt{Eq}\ A\ x\ y\text{)} \\
\texttt{refl} & : & \Pi A : \mathsf{Set} \,.\, \Pi x : A \,.\; x =_A x \\
\texttt{eq_elim} & : & \Pi A : \mathsf{Set} \,.\; \Pi P : A \to \mathsf{Set} \,.\; \Pi x, y : A \,. \\
 & & \qquad P\,x \;\to\; x =_A y \;\to\; P\,y
\end{array}$$

δ-regla asociada

$$\texttt{eq\_elim}\ A\ P\ x\ x\ p\ (\texttt{refl}\ A\ x) \;\succ\; p$$

Obs.: La δ-regla implementa la reducción usual del corte de $=$:

$$\dfrac{\begin{array}{cc} \vdots & \\ p : P\,x & \dfrac{}{\texttt{refl}\ A\ x : x =_A x} \end{array}}{\texttt{eq\_elim}\ A\ P\ x\ x\ p\ (\texttt{refl}\ A\ x) : P\,x} \;\succ\; \begin{array}{c} \vdots \\ p : P\,x \end{array}$$

El tipo identidad $x =_A y$ (2/3)

El tipo identidad $x =_A y$ cumple las propiedades deseadas:
(Usando la notación $\forall x : A \,.\, B \;\equiv\; \Pi x : A \,.\, B$)

- **Lema:** $\forall A : \mathsf{Set} \,.\; \forall x : A \,.\; x =_A x$

  Prueba: `refl`

- **Lema:** $\forall A : \mathsf{Set} \,.\; \forall x, y : A \,.\; x =_A y \;\to\; y =_A x$

  Prueba: $\lambda A : \mathsf{Set} \,.\, \lambda x, y : A \,.\, \lambda e : (x =_A y) \,.$
  $\quad$ `eq_elim` $A\ (\lambda z : A \,.\, z =_A x)\ x\ y\ (\mathtt{refl}\ A\ x)\ e$

- **Lema:** $\forall A : \mathsf{Set} \,.\; \forall x, y, z : A \,.\; x =_A y \;\to\; y =_A z \;\to\; x =_A z$

  Prueba: $\lambda A : \mathsf{Set} \,.\, \lambda x, y, z : A \,.\, \lambda e : (x =_A y) \,.\, \lambda e' : (y =_A z) \,.$
  $\quad$ `eq_elim` $A\ (\lambda h : A \,.\, x =_A h)\ y\ z\ e\ e'$

- **Lema:** $\forall A : \mathsf{Set} \,.\; \forall x, y : A \,.\; x =_A y \;\to$
  $\quad \forall B : \mathsf{Set} \,.\; \forall f : A \to B \,.\; f\ x =_B f\ y$

  Prueba: $\lambda A : \mathsf{Set} \,.\, \lambda x, y : A \,.\, \lambda e : (x =_A y) \,.\, \lambda B : \mathsf{Set} \,.\, \lambda f : A \to B \,.$
  $\quad$ `eq_elim` $A\ (\lambda z : A \,.\, f\ x =_B f\ z)\ x\ y\ (\mathtt{refl}\ B\ (f\ x))$

El tipo identidad $x =_A y$ (3/3)

También se pueden demostrar los restantes axiomas de Peano:
(Ya demostramos el principio de inducción: `nat_rec`)

- **Lema:** $\forall x : \mathtt{Nat}\,.\ \mathtt{plus}\ x\ \mathsf{0} =_{\mathtt{Nat}} x$

  Prueba: $\lambda x : \mathtt{Nat}\,.\,\mathtt{refl}\ \mathtt{Nat}\ x$ (por conversión)

- **Lema:** $\forall x, y : \mathtt{Nat}\,.\ \mathtt{plus}\ x\ (\mathtt{S}\ y) =_{\mathtt{Nat}} \mathtt{S}\ (\mathtt{plus}\ x\ y)$

  Prueba: $\lambda x, y : \mathtt{Nat}\,.\,\mathtt{refl}\ \mathtt{Nat}\ (\mathtt{S}\ (\mathtt{plus}\ x\ y))$ (por conversión)

- Axiomas de `mult`: análogo

- **Lema:** $\forall x, y : \mathtt{Nat}\,.\ \mathtt{S}\ x =_{\mathtt{Nat}} \mathtt{S}\ y \ \rightarrow\ x =_{\mathtt{Nat}} y$

  Prueba: $\lambda x, y : \mathtt{Nat}\,.\,\lambda e : (\mathtt{S}\ x =_{\mathtt{Nat}} \mathtt{S}\ y)\,.$
  $\mathtt{eq\_elim}\ \mathtt{Nat}\ (\lambda z : \mathtt{Nat}\,.\,\mathtt{pred}\ (\mathtt{S}\ x) =_{\mathtt{Nat}} \mathtt{pred}\ z)$
  $(\mathtt{S}\ x)\ (\mathtt{S}\ y)\ (\mathtt{refl}\ \mathtt{Nat}\ x)\ e$

- **Lema:** $\forall x : \mathtt{Nat}\,.\ \mathtt{S}\ x =_{\mathtt{Nat}} \mathsf{0} \ \rightarrow\ \bot$

  Prueba: Véase más adelante

Eliminación fuerte (1/2)

- `nat_rec` permite iterar una función sobre datos... $n \mapsto 2^n$
  ... pero no permite iterar una función sobre tipos $n \mapsto \mathtt{Nat}^n$

Principio de eliminación fuerte ($+ \delta$-reglas)

$$\mathtt{nat\_rect} \;:\; \mathsf{Set} \to (\mathtt{Nat} \to \mathsf{Set} \to \mathsf{Set}) \to \mathtt{Nat} \to \mathsf{Set}$$

$$\begin{array}{lcl} \mathtt{nat\_rect}\, X\, F\, \mathtt{0} & \succ & X \\ \mathtt{nat\_rect}\, X\, F\, (\mathtt{S}\, n) & \succ & F\, n\, (\mathtt{nat\_rect}\, X\, F\, n) \end{array}$$

- **Ejemplo:**

$$\begin{array}{lcl} \mathtt{marrow} & : & \mathtt{Nat} \to \mathsf{Set} \to \mathsf{Set} \to \mathsf{Set} \\ & :\equiv & \lambda n : \mathtt{Nat}\,.\, \lambda A, B : \mathsf{Set}\,. \\ & & \quad \mathtt{nat\_rect}\ B\ (\lambda \_ : \mathtt{Nat}\,.\, \lambda X : \mathsf{Set}\,.\, A \to X) \end{array}$$

Para todos $A, B : \mathsf{Set}$ y $n \in \mathbb{N}$, tenemos que:

$$\mathtt{marrow}\ (\mathtt{S}^n\ \mathtt{0})\ A\ B \;\succ^*\; \underbrace{A \to \cdots \to A}_{n} \to B$$

Eliminación fuerte (2/2)

- **Otro ejemplo:**

$$\begin{array}{lcl} \texttt{Null} & : & \texttt{Nat} \to \text{Set} \\ & :\equiv & \lambda n : \texttt{Nat} \,.\, \texttt{nat\_rect}\ \mathbf{1}\ (\lambda \_ : \texttt{Nat} \,.\, \lambda \_ : \text{Set} \,.\, \mathbf{0})\ n \end{array}$$

Por construcción:
$$\begin{array}{lcl} \texttt{Null}\ 0 & \succ^* & \mathbf{1} \\ \texttt{Null}\ (\texttt{S}\ n) & \succ^* & \mathbf{0} \end{array}$$

- Permite derivar el axioma de Peano:

Lema: $\forall x : \texttt{Nat} \,.\ \texttt{S}\ x =_{\texttt{Nat}} 0 \ \to\ \bot$

Prueba: $\lambda x : \texttt{Nat} \,.\, \lambda e : (\texttt{S}\ x =_{\texttt{Nat}} 0) \,.$
$\texttt{eq\_elim}\ \texttt{Nat}\ \texttt{Null}\ 0\ (\texttt{S}\ x)\ \langle\rangle\ (\texttt{eq\_sym}\ \texttt{Nat}\ (\texttt{S}\ x)\ 0\ e)$

- Por todo lo anterior:

Proposición (Inclusión HA $\subset$ MLTT)

La teoría de tipos de Martin-Löf es una extensión (no conservativa) de la aritmética de Heyting: HA $\subset$ MLTT

Obs.: Extensión no conservativa pues MLTT $\vdash$ Cons(HA) (Ejercicio)

Plan

Expresividad de MLTT

- Como lenguaje de programación, MLTT contiene mucho más que el sistema T de Gödel (pero no contiene el sistema F de Girard)
- En particular, MLTT permite el polimorfismo predicativo
- Como sistema lógico, MLTT contiene mucho más que la Aritmética intuicionista de los tipos finitos HA^ω [(1)] (pero no contiene HA2)
- Aunque no contenga HA2, MLTT permite expresar cuantificaciones universales sobre predicados de tipo $\mathtt{Nat} \to \mathrm{Set}$ $(: \mathrm{Type})$, que definen una noción predicativa de conjunto [(2)]
- Finalmente, la identificación entre tipos y proposiciones permite usar los programas como pruebas, y las pruebas como programas

---

[(1)] HA^ω = extensión conservativa de HA cuyos tipos y términos son los del sistema T, y cuyo lenguaje de fórmulas tiene cuantificaciones para todos los tipos del sistema T

[(2)] Las fórmulas (: Set) que sirven para construir tales conjuntos (por comprensión) prohiben cuantificaciones sobre los conjuntos (que inducen tipos grandes : Type)

Observación sobre la extensionalidad

- MLTT provee una noción de igualdad $x =_A y$ $(\equiv \ \mathtt{Eq}\ A\ x\ y)$ para cada tipo $A : \mathsf{Set}$, inclusive para los tipos funcionales

- Es obvio que dos funciones iguales son extensionalmente iguales:

$$\forall A, B : \mathsf{Set}\,.\ \forall f, g : A \to B\,.\ f =_{A\to B} g \ \to\ \forall x : A\,.\ f\ x =_B g\ x$$

Prueba:

$$\lambda A, B : \mathsf{Set}\,.\,\lambda f, g : A \to B\,.\,\lambda e : (f =_{A\to B} g)\,.\,\lambda x : A\,.\ \mathtt{eq\_elim}\ (A \to B)\ (\lambda h : A \to B\,.\, f\ x =_B h\ x)\ f\ g\ (\mathtt{refl}\ B\ (f\ x))\ e$$

- Pero MLTT no cumple el recíproco (= axioma de extensionalidad):

$$\mathtt{MLTT} \ \not\vdash \ (\forall x : A\,.\ f\ x =_B g\ x) \ \to\ f =_{A\to B} g$$

$\Rightarrow$ La igualdad de MLTT es una igualdad intensional

- **Intuición:** $\forall l : \mathtt{List\ Nat}\,.\ \mathtt{quick\_sort}\ l = \mathtt{bubble\_sort}\ l$

  pero $\mathtt{quick\_sort} \neq \mathtt{bubble\_sort}$

Ejemplo de formalización aritmética (1/4)

«En lo siguiente, los objetos matemáticos serán considerados como nuestras propias construcciones»

— Per Martin-Löf, *A Theory of Types*, 1971

- **Objectivo:** Demostrar la fórmula:

$$\forall x : \mathtt{Nat}\,.\ \exists y : \mathtt{Nat}\,.\quad x = 2 \times y \ \vee\ x = 2 \times y + 1$$

Es decir: Construir un término de tipo:

$$\Pi x : \mathtt{Nat}\,.\ \Sigma y : \mathtt{Nat}\,.\quad x =_{\mathtt{Nat}} \mathtt{mult}\ 2\ y \ +\ x =_{\mathtt{Nat}} \mathtt{S}\,(\mathtt{mult}\ 2\ y)$$

- En lo siguiente, se escriben:

$$\begin{aligned} D_0[x,\,y] &:\equiv x =_{\mathtt{Nat}} \mathtt{mult}\ 2\ y \\ D_1[x,\,y] &:\equiv x =_{\mathtt{Nat}} \mathtt{S}\,(\mathtt{mult}\ 2\ y) \\ D[x,\,y] &:\equiv D_0[x,\,y] \vee D_1[x,\,y] \\ E[x] &:\equiv \exists y : \mathtt{Nat}\,.\ D[x,y] \end{aligned}$$

Ejemplo de formalización aritmética (2/4)

$$\begin{array}{rcl@{\qquad}rcl} D_0[x,\,y] & :\equiv & x =_{\mathtt{Nat}} \mathtt{mult}\ \mathtt{2}\ y & D[x,\,y] & :\equiv & D_0[x,\,y] \vee D_1[x,\,y] \\ D_1[x,\,y] & :\equiv & x =_{\mathtt{Nat}} \mathtt{S}\,(\mathtt{mult}\ \mathtt{2}\ y) & E[x] & :\equiv & \exists y : \mathtt{Nat}\,.\ D[x,\,y] \end{array}$$

- **Lema** $\mathtt{ind\_base} : E[\mathtt{0}]$

$$\begin{aligned} :=\ & \mathtt{dpair}\ \mathtt{Nat}\ (\lambda y : \mathtt{Nat}\,.\, D[\mathtt{0},\,y])\ \mathtt{0} \\ & \quad (\mathtt{inl}\ D_0[\mathtt{0},\,\mathtt{0}]\ D_1[\mathtt{0},\,\mathtt{0}]\ (\mathtt{refl}\ \mathtt{Nat}\ \mathtt{0})) \end{aligned}$$

- **Lema** $\mathtt{ind\_step}_0 : \forall x, y : \mathtt{Nat}\,.\, D_0[x,\,y] \to D_1[\mathtt{S}\,x,\,y]$

$$\begin{aligned} :=\ & \lambda x, y : \mathtt{Nat}\,.\, \lambda h : D_0[x,\,y]\,. \\ & \quad \mathtt{eq\_elim}\ \mathtt{Nat}\ (\lambda z : \mathtt{Nat}\,.\, \mathtt{S}\,x =_{\mathtt{Nat}} \mathtt{S}\,z) \\ & \qquad x\ (\mathtt{mult}\ \mathtt{2}\ y)\ (\mathtt{refl}\ \mathtt{Nat}\ (\mathtt{S}\,x))\ h \end{aligned}$$

- **Lema** $\mathtt{ind\_step}'_0 : \forall x, y : \mathtt{Nat}\,.\, D_0[x,\,y] \to E[\mathtt{S}\,x]$

$$\begin{aligned} :=\ & \lambda x, y : \mathtt{Nat}\,.\, \lambda h : D_0[x,\,y]\,. \\ & \quad \mathtt{dpair}\ \mathtt{Nat}\ (\lambda z : \mathtt{Nat}\,.\, D[\mathtt{S}\,x,\,z])\ y \\ & \qquad (\mathtt{inr}\ D_0[\mathtt{S}\,x,\,y]\ D_1[\mathtt{S}\,x,\,y]\ (\mathtt{ind\_step}_0\ x\ y\ h)) \end{aligned}$$

Ejemplo de formalización aritmética (3/4)

$$
\begin{array}{rcl@{\qquad}rcl}
D_0[x,\,y] & :\equiv & x =_{\mathtt{Nat}} \mathtt{mult}\ \mathtt{2}\ y & D[x,\,y] & :\equiv & D_0[x,\,y] \vee D_1[x,\,y] \\
D_1[x,\,y] & :\equiv & x =_{\mathtt{Nat}} \mathtt{S}\,(\mathtt{mult}\ \mathtt{2}\ y) & E[x] & :\equiv & \exists y : \mathtt{Nat}\,.\ D[x,\,y]
\end{array}
$$

- **Lema** $\mathtt{ind\_step}_1 \ : \ \forall x, y : \mathtt{Nat}\,.\, D_1[x,\,y] \to D_0[\mathtt{S}\,x,\,\mathtt{S}\,y]$

$:=$ $\lambda x, y : \mathtt{Nat}\,.\,\lambda h : D_1[x,\,y]\,.$
$\mathtt{eq\_elim}\ \mathtt{Nat}\ (\lambda z : \mathtt{Nat}\,.\,\mathtt{S}\,x =_{\mathtt{Nat}} \mathtt{S}\,z)$
$x\ (\mathtt{S}\,(\mathtt{mult}\ \mathtt{2}\ y))\ (\mathtt{refl}\ \mathtt{Nat}\ (\mathtt{S}\,x))\ h$

- **Lema** $\mathtt{ind\_step}'_1 \ : \ \forall x, y : \mathtt{Nat}\,.\, D_1[x,\,y] \to E[\mathtt{S}\,x]$

$:=$ $\lambda x, y : \mathtt{Nat}\,.\,\lambda h : D_0[x,\,y]\,.$
$\mathtt{dpair}\ \mathtt{Nat}\ (\lambda z : \mathtt{Nat}\,.\,D[\mathtt{S}\,x,\,z])\ (\mathtt{S}\,y)$
$(\mathtt{inl}\ D_0[\mathtt{S}\,x,\,\mathtt{S}\,y]\ D_1[\mathtt{S}\,x,\,\mathtt{S}\,y]\ (\mathtt{ind\_step}_1\ x\ y\ h))$

- **Lema** $\mathtt{ind\_step} \ : \ \forall x : \mathtt{Nat}\,.\, E[x] \to E[\mathtt{S}\ x]$

$:=$ $\lambda x : \mathtt{Nat}\,.\,\lambda h : E[x]\,.$
$\mathtt{dsum\_elim}\ \mathtt{Nat}\ (\lambda x : \mathtt{Nat}\,.\,D[x])\ E[\mathtt{S}\ x]$
$(\lambda x : \mathtt{Nat}\,.\,\mathtt{sum\_elim}\ D_0[x]\ D_1[x]\ E[\mathtt{S}\ x]$
$(\mathtt{ind\_step}'_0\ x)\ (\mathtt{ind\_step}'_1\ x))\ h$

Ejemplo de formalización aritmética (4/4)

$$\begin{array}{rcl} D_0[x,\,y] & :\equiv & x =_{\mathtt{Nat}} \mathtt{mult}\ 2\ y \\ D_1[x,\,y] & :\equiv & x =_{\mathtt{Nat}} \mathtt{S}\,(\mathtt{mult}\ 2\ y) \end{array} \qquad \begin{array}{rcl} D[x,\,y] & :\equiv & D_0[x,\,y] \vee D_1[x,\,y] \\ E[x] & :\equiv & \exists y : \mathtt{Nat}\,.\ D[x,\,y] \end{array}$$

- Ya demostramos:
  - **Lema** `ind_base` : $E[0]$
  - **Lema** `ind_step` : $\forall x : \mathtt{Nat}\,.\, E[x] \to E[\mathtt{S}\,x]$

  y por lo tanto:

- **Teorema** `foo` :

$$\forall x : \mathtt{Nat}\,.\ \underbrace{\exists y : \mathtt{Nat}\,.\ \ x =_{\mathtt{Nat}} \mathtt{mult}\ 2\ y \ \vee\ x =_{\mathtt{Nat}} \mathtt{S}\,(\mathtt{mult}\ 2\ y)}_{E[x]}$$

  $:=\quad \lambda x : \mathtt{Nat}\,.\ \mathtt{nat\_rec}\ (\lambda z : \mathtt{Nat}\,.\, E[z])\ \mathtt{ind\_base}\ \mathtt{ind\_step}\ x$

- ¿Cómo extraer la función `div2` : $\mathtt{Nat} \to \mathtt{Nat}$ subyacente?

Extracción de programas (1/3)

- **Recordatorio:** $\exists x : A \,.\, B\, x \;\equiv\; \Sigma x : A \,.\, B\, x \;\equiv\; \texttt{DSum}\ A\ B$, donde:

$$\texttt{DSum} \;:\; \forall A : \mathsf{Set} \,.\, (A \to \mathsf{Set}) \to \mathsf{Set} \qquad (\exists x : A \,.\, B\, x \;:\equiv\; \texttt{DSum}\ A\ B)$$

$$\texttt{dpair} \;:\; \forall A : \mathsf{Set} \,.\, \forall B : A \to \mathsf{Set} \,.\, \forall x : A \,.\, B\, x \to \exists x : A \,.\, B\, x \qquad (\langle a, b\rangle \;:\equiv\; \texttt{dpair}\ A\ B\ a\ b)$$

$$\texttt{dsum\_rec} \;:\; \forall A : \mathsf{Set} \,.\, \forall B : A \to \mathsf{Set} \,.\, \forall X : (\exists x : A \,.\, B) \to \mathsf{Set} \,.\, (\forall x : A \,.\, \forall y : B\, x \,.\; X\, \langle x, y\rangle) \to \forall s : (\exists x : A \,.\, B\, x) \,.\, X\, s$$

$$\texttt{dsum\_rec}\ A\ B\ X\ f\ (\texttt{dpair}\ A\ B\ a\ b) \;\succ\; f\ a\ b$$

- A partir del recursor «`dsum_rec`», se pueden construir las proyecciones dependientes (ejercicio):

$$\texttt{dfst} \;:\; \forall A : \mathsf{Set} \,.\, \forall B : A \to \mathsf{Set} \,.\, (\exists x : A \,.\, B\, x) \;\to\; A$$

$$\texttt{dsnd} \;:\; \forall A : \mathsf{Set} \,.\, \forall B : A \to \mathsf{Set} \,.\, \forall s : (\exists x : A \,.\, B\, x) \,.\; B\ (\texttt{dfst}\ A\ B\ s)$$

de tal modo que:

$$\texttt{dfst}\ A\ B\ (\texttt{dpair}\ A\ B\ a\ b) \;\succ^*\; a$$
$$\texttt{dsnd}\ A\ B\ (\texttt{dpair}\ A\ B\ a\ b) \;\succ^*\; b$$

Extracción de programas (2/3)

A partir de los términos:

$$\texttt{foo} \;:\; \forall x:\texttt{Nat}.\ \exists y:\texttt{Nat}.\ \underbrace{x = \texttt{mult 2}\ y \ \vee\ x = \texttt{S}\,(\texttt{mult 2}\ y)}_{D[x,y]}$$

$$\texttt{dfst} \;:\; \forall A:\mathsf{Set}.\forall B:A\to\mathsf{Set}.\,(\exists x:A.\,B\,x)\ \to\ A$$

$$\texttt{dsnd} \;:\; \forall A:\mathsf{Set}.\forall B:A\to\mathsf{Set}.\forall s:(\exists x:A.\,B\,x).\ B\ (\texttt{dfst}\ A\ B\ s)$$

se pueden construir los términos:

$$\begin{aligned}\texttt{div2} &:\equiv \lambda x:\texttt{Nat}.\,\texttt{dfst Nat}\ (\lambda y:\texttt{Nat}.\,D[x,y])\ (\texttt{foo}\ x)\\ &: \texttt{Nat}\to\texttt{Nat}\end{aligned}$$

$$\begin{aligned}\texttt{div2\_correct} &:\equiv \lambda x:\texttt{Nat}.\,\texttt{dsnd Nat}\ (\lambda y:\texttt{Nat}.\,D[x,y])\ (\texttt{foo}\ x)\\ &: \forall x:\texttt{Nat}.\ D[x,\ \texttt{dfst}\ (\lambda y:\texttt{Nat}.\,D[x,y])\ (\texttt{foo}\ x)]\\ &: \forall x:\texttt{Nat}.\ D[x,\ \texttt{div2}\ x] && \text{(por conversión)}\\ &: \forall x:\texttt{Nat}.\ x = \texttt{mult 2}\ (\texttt{div2}\ x)\ \vee\ x = \texttt{S}\,(\texttt{mult 2}\ (\texttt{div2}\ x))\end{aligned}$$

Extracción de programas (3/3)

Un mecanismo muy general: Dados tipos $A, B : \mathrm{Set}$, un predicado $C : A \to B \to \mathrm{Set}$ y un término (de prueba)

$$M \quad : \quad \forall x : A \,.\; \exists y : B \,.\; C\ x\ y$$

siempre se pueden construir los términos:

$$\begin{array}{rcll} f & :\equiv & \lambda x : A \,.\, \mathtt{dfst}\ B\ (C\ x)\ (M\ x) & : \quad A \to B \\ f_{\mathsf{correct}} & :\equiv & \lambda x : A \,.\, \mathtt{dsnd}\ B\ (C\ x)\ (M\ x) & : \quad \forall x : A \,.\; C\ x\ (f\ x) \end{array}$$

Intuitivamente:

- f recoge los testigos de $\exists y : B \,.\; C\ x\ y$ (para cada $x : A$)
- f_{correct} recoge las correspondientes justificaciones

Todo esto funciona sin suponer la unicidad de $y : B$...
... ¿un sabor de axioma de elección?

El axioma de elección intensional

Teorema (Axioma de elección intensional)

$$\forall A, B : \mathsf{Set}\,.\ \forall C : (A \to B \to \mathsf{Set})\,.$$
$$(\forall x : A\,.\ \exists y : B\,.\ C\ x\ y)\ \to\ \exists f : (A \to B)\,.\ \forall x : A\,.\ C\ x\ (f\ x)$$

Prueba: $\lambda A, B : \mathsf{Set}\,.\,\lambda C : (A \to B \to \mathsf{Set})\,.\,\lambda h : (\forall x : A\,.\,\exists y : B\,.\ C\ x\ y)\,.$
$\mathtt{dpair}\ (A \to B)\ (\lambda f : (A \to B)\,.\,\forall x : A\,.\ C\ x\ (f\ x))$
$(\lambda x : A\,.\,\mathtt{dfst}\ B\ (C\ x)\ (h\ x))\ (\lambda x : A\,.\,\mathtt{dsnd}\ B\ (C\ x)\ (h\ x))$

Axioma de elección intensional – Variante

Toda función sobreyectiva tiene inversa por la derecha:

$$\forall A, B : \mathsf{Set}\,.\ \forall f : (A \to B)\,.$$
$$(\forall y : B\,.\ \exists x : A\,.\ f\ x =_B y)\ \to\ \exists g : (B \to A)\,.\ \forall y : B\,.\ f\ (g\ y) =_B y$$

Prueba: Ejercicio.

¡Cuidado! El axioma de elección de MLTT (intensional e intuicionista) es mucho más débil que el axioma de elección de ZF (extensional y clásico). En particular, no implica ni el lema de Zorn, ni el teorema de Zermelo

Plan

Una teoría de tipos de Martin-Löf (1/2)

En esta sección, se considera la teoría de tipos de Martin-Löf (MLTT) definida a partir de las siguientes constantes y δ-reglas:

$$\begin{array}{rcl} \mathbf{0} & : & \mathsf{Set} \\ \texttt{empty\_rec} & : & \Pi X : \mathbf{0} \to \mathsf{Set} \,.\, \Pi x : \mathbf{0} \,.\, X\ x \end{array}$$

$$\begin{array}{rcl} \mathbf{1} & : & \mathsf{Set} \\ \langle\rangle & : & \mathbf{1} \\ \texttt{unit\_rec} & : & \Pi X : \mathbf{1} \to \mathsf{Set} \,.\, X\ \langle\rangle \to \Pi x : \mathbf{1} \,.\, X\ x \\ \texttt{unit\_rec}\ X\ x\ \langle\rangle & \succ & x \end{array}$$

$$\begin{array}{rcl} \texttt{Bool} & : & \mathsf{Set} \\ \texttt{true} & : & \texttt{Bool} \\ \texttt{false} & : & \texttt{Bool} \\ \texttt{bool\_rec} & : & \Pi X : \texttt{Bool} \to \mathsf{Set} \,.\, X\ \texttt{true} \to X\ \texttt{false} \to \Pi x : \texttt{Bool} \,.\, X\ x \\ \texttt{bool\_rec}\ X\ x\ y\ \texttt{true} & \succ & x \\ \texttt{bool\_rec}\ X\ x\ y\ \texttt{false} & \succ & y \end{array}$$

$$\begin{array}{rcl} \texttt{Nat} & : & \mathsf{Set} \\ \texttt{0} & : & \texttt{Nat} \\ \texttt{S} & : & \texttt{Nat} \to \texttt{Nat} \\ \texttt{nat\_rec} & : & \Pi X : \texttt{Nat} \to \mathsf{Set} \,.\, X\ \texttt{0} \to (\Pi y : \texttt{Nat} \,.\, X\ y \to X\ (\texttt{S}\ y)) \to \Pi x : \texttt{Nat} \,.\, X\ x \\ \texttt{nat\_rec}\ X\ x\ f\ \texttt{0} & \succ & x \\ \texttt{nat\_rec}\ X\ x\ f\ (\texttt{S}\ n) & \succ & f\ n\ (\texttt{nat\_rec}\ X\ x\ f\ n) \\ \texttt{nat\_rect} & : & \mathsf{Set} \to (\texttt{Nat} \to \mathsf{Set} \to \mathsf{Set}) \to \texttt{Nat} \to \mathsf{Set} \\ \texttt{nat\_rect}\ X\ F\ \texttt{0} & \succ & x \\ \texttt{nat\_rect}\ X\ F\ (\texttt{S}\ n) & \succ & F\ n\ (\texttt{nat\_rect}\ X\ F\ n) \end{array}$$

$(\cdots)$

Una teoría de tipos de Martin-Löf (2/2)

$$
\begin{array}{rcl}
\texttt{Prod} & : & \mathsf{Set} \to \mathsf{Set} \to \mathsf{Set} \\
\texttt{pair} & : & \Pi A, B : \mathsf{Set} \,.\, A \to B \to \texttt{Prod}\ A\ B \\
\texttt{prod\_rec} & : & \Pi A, B : \mathsf{Set} \,.\, \Pi X : \texttt{Prod}\ A\ B \to \mathsf{Set} \,. \\
 & & \quad (\Pi x : A \,.\, \Pi y : B \,.\, X\ (\texttt{pair}\ A\ B\ x\ y)) \to \Pi z : \texttt{Prod}\ A\ B \,.\, X\ z \\
\multicolumn{3}{l}{\texttt{prod\_rec}\ A\ B\ X\ f\ (\texttt{pair}\ A\ B\ a\ b) \quad \succ \quad f\ a\ b}
\end{array}
$$

$$
\begin{array}{rcl}
\texttt{Sum} & : & \mathsf{Set} \to \mathsf{Set} \to \mathsf{Set} \\
\texttt{inl} & : & \Pi A, B : \mathsf{Set} \,.\, A \to \texttt{Sum}\ A\ B \\
\texttt{inr} & : & \Pi A, B : \mathsf{Set} \,.\, B \to \texttt{Sum}\ A\ B \\
\texttt{sum\_rec} & : & \Pi A, B : \mathsf{Set} \,.\, \Pi X : \texttt{Sum}\ A\ B \to \mathsf{Set} \,. \\
 & & \quad (\Pi x : A \,.\, X\ (\texttt{inl}\ A\ B\ x)) \to (\Pi y : B \,.\, X\ (\texttt{inr}\ A\ B\ y)) \to \Pi z : \texttt{Sum}\ A\ B \,.\, X\ z \\
\multicolumn{3}{l}{\texttt{sum\_rec}\ A\ B\ X\ f\ g\ (\texttt{inl}\ A\ B\ a) \quad \succ \quad f\ a} \\
\multicolumn{3}{l}{\texttt{sum\_rec}\ A\ B\ X\ f\ g\ (\texttt{inr}\ A\ B\ b) \quad \succ \quad g\ b}
\end{array}
$$

$$
\begin{array}{rcl}
\texttt{DSum} & : & \Pi A : \mathsf{Set} \,.\, (A \to \mathsf{Set}) \to \mathsf{Set} \\
\texttt{pair} & : & \Pi A : \mathsf{Set} \,.\, \Pi B : A \to \mathsf{Set} \,.\, \Pi x : A \,.\, B\ x \to \texttt{DSum}\ A\ B \\
\texttt{dsum\_rec} & : & \Pi A : \mathsf{Set} \,.\, \Pi B : A \to \mathsf{Set} \,.\, \Pi X : \texttt{DSum}\ A\ B \to \mathsf{Set} \,. \\
 & & \quad (\Pi x : A \,.\, \Pi y : B\ x \,.\, X\ (\texttt{dpair}\ A\ B\ x\ y)) \to \Pi z : \texttt{DSum}\ A\ B \,.\, X\ z \\
\multicolumn{3}{l}{\texttt{dsum\_rec}\ A\ B\ X\ f\ (\texttt{dpair}\ A\ B\ a\ b) \quad \succ \quad f\ a\ b}
\end{array}
$$

$$
\begin{array}{rcl}
\texttt{Eq} & : & \Pi A : \mathsf{Set} \,.\, A \to A \to \mathsf{Set} \\
\texttt{refl} & : & \Pi A : \mathsf{Set} \,.\, \Pi x : A \,.\, \texttt{Eq}\ A\ x\ x \\
\texttt{eq\_elim} & : & \Pi A : \mathsf{Set} \,.\, \Pi P : A \to \mathsf{Set} \,.\, \Pi x, y : A \,.\, P\ x \to \texttt{Eq}\ A\ x\ y \to P\ y \\
\multicolumn{3}{l}{\texttt{eq\_elim}\ A\ P\ x\ x\ p\ (\texttt{refl}\ A\ x) \quad \succ \quad p}
\end{array}
$$

Ese sistema es confluente y cumple la *subject reduction* (Ejercicio)

Formas canónicas (1/3)

Notaciones: Tres especies de constantes:

- *ind* = constante inductiva (`Nat`, `Prod`, `Eq`, etc.)
- *constr* = constructor (`0`, `pair`, `refl`, etc.)
- *destr* = destructor (`nat_rec`, `prod_rec`, `eq_elim`, etc.)

Cada constante c tiene una aridad $\sharp c$ (dada por su tipo)

Por ejemplo: $\sharp\texttt{Eq} = 3$, $\sharp\texttt{refl} = 2$, $\sharp\texttt{eq\_elim} = 6$

Definición (Forma canónica)

Una forma canónica es todo término de la forma:

- Set, Type, $\Pi x : A \,.\, B$, *ind* $\vec{N}$ (Tipos o familias de tipos)
- $\lambda x : A \,.\, M$, *constr* $\vec{N}$ (Funciones y constructores aplicados)
- *destr* $N_1 \cdots N_k$, con $k < \sharp$*destr* (Destructores parcialmente aplicados)

Obs.: Las formas canónicas son estables por reducción y por sustitución (Ejercicio)

Formas canónicas (2/3)

Proposición (Formas normales)

En el contexto vacío, todo término M bien tipado y en forma normal es una forma canónica. Además:

1. Si $M : \text{Set}$, entonces M es
   - o bien de la forma $M \equiv \Pi x : A \,.\, B$
   - o bien de la forma $M \equiv \mathit{ind}\ P_1 \cdots P_n$ (con $n = \sharp \mathit{ind}$)
2. Si $M : \Pi x : A \,.\, B$, entonces M es
   - o bien de la forma $M \equiv \lambda x : A' \,.\, M$
   - o bien de la forma $M \equiv \mathit{destr}\ N_1 \cdots N_k$ (con $k < \sharp \mathit{destr}$)
3. Si $M : \mathit{ind}\ P_1 \cdots P_n$, entonces M es de la forma
   $$M \equiv \mathit{constr}\ N_1 \cdots N_k \quad \text{(con } k = \sharp \mathit{constr}\text{)}$$
   donde *constr* es uno de los constructores de *ind*

Demostración. Ejercicio

Formas canónicas (3/3)

Corolario (Formas canónicas de los tipos inductivos)

En el contexto vacío:

- Todo término $M : \mathtt{Nat}$ en forma normal es de la forma $M \equiv \mathtt{S}^n\ \mathtt{0}$ (con $n \in \mathbb{N}$)
- Los únicos términos de tipo `Bool` en forma normal son `true` y `false`
- El único término de tipo $\mathbf{1}$ en forma normal es $\langle\rangle$
- No existe ningún término de tipo $\mathbf{0}$ en forma normal
- Todo término $M : \mathtt{Prod}\, A\, B\ (= A \times B)$ en forma normal es de la forma $M \equiv \mathtt{pair}\, A'\, B'\, a\, b$, con $A' \equiv {\downarrow}A$, $B' \equiv {\downarrow}B$, $a : A$ y $b : B$
- Todo término $M : \mathtt{Sum}\, A\, B\ (= A + B)$ en forma normal es de la forma:
  - $M \equiv \mathtt{inl}\, A'\, B'\, a$, con $A' \equiv {\downarrow}A$, $B' \equiv {\downarrow}A$ y $a : A$, o bien
  - $M \equiv \mathtt{inr}\, A'\, B'\, b$, con $A' \equiv {\downarrow}A$, $B' \equiv {\downarrow}A$ y $b : B$
- Todo término $M : \mathtt{DSum}\, A\, B\ (= \Sigma x : A\,.\, B\, x)$ en forma normal es de la forma $M \equiv \mathtt{dpair}\, A'\, B'\, a\, b$, con $A' \equiv {\downarrow}A$, $B' \equiv {\downarrow}B$, $a : A$ y $b : B\, a$
- Todo término $M : \mathtt{Eq}\, A\, a_1\, a_2\ (\equiv a_1 =_A a_2)$ en forma normal es de la forma $M \equiv \mathtt{refl}\, A'\, a'$, con $A' = {\downarrow}A$ y $a' \equiv {\downarrow}a_1 \equiv {\downarrow}a_2$.

  En particular, la existencia de M implica que $a_1 \cong a_2$

Demostración. Ejercicio

El teorema de normalización fuerte

Los resultados anteriores (sobre las formas canónicas) tienen pruebas puramente combinatorias (i.e. en HA). No es el caso del siguiente

Teorema (Normalización fuerte)

Si $\Gamma \vdash M : A$, entonces M es fuertemente normalizante

Demostración. Véase próximas diapositivas

- **Obs.:** Por un lema anterior, sabemos que $\Gamma \vdash M : A$ implica que $A \equiv \mathsf{Type}$ o $\Gamma \vdash A : s$ para algún $s \in \{\mathsf{Set}, \mathsf{Type}\}$

  Aplicando el teorema de normalización fuerte al juicio $\Gamma \vdash A : s$, también se deduce que A es fuertemente normalizante

- De modo análogo, se demuestra que $\vdash \Gamma$ context implica que todos los tipos en Γ son fuertemente normalizantes

Consecuencias del teorema de normalización fuerte

El teorema de normalización fuerte se cumple en todos los contextos (bien formados). Además:

Corolarios

En el contexto vacío:

1. **Consistencia:** No existe ningún término $M : \mathbf{0}$
2. **Disyunción:** Todo término $M : \mathtt{Sum}\, A\, B\ (\equiv A \vee B)$ se reduce:
   - o bien sobre $\mathtt{inl}\ A'\ B'\ a$, con $A' \equiv \downarrow A$, $B' \equiv \downarrow B$ y $a : A$
   - o bien sobre $\mathtt{inr}\ A'\ B'\ b$, con $A' \equiv \downarrow A$, $B' \equiv \downarrow B$ y $b : B$
3. **Existencia:** Todo término $M : \mathtt{DSum}\, A\, B\ (\equiv \exists x : A\,.\, B\, x)$ se reduce sobre $\mathtt{dpair}\ A'\ B'\ a\ b$, con $A' \equiv \downarrow A$, $B' \equiv \downarrow B$, $a : A$ (testigo) y $b : B\, a$ (justificación)
4. **Igualdad:** Existe un término de tipo $\mathtt{Eq}\, A\, a_1\, a_2\ (\equiv a_1 =_A a_2)$ si y sólo si los términos $a_1, a_2 : A$ son convertibles: $a_1 \cong a_2$

Demostración. Ejercicio

Candidatos de reducibilidad

- $\mathbf{SN}$ = conjunto de los términos fuertemente normalizantes
- $\mathrm{Red}_1(M) := \{M' \in \Lambda : M \succ M'\}$
- Un término neutro es un término que no está en forma canonica

Definición (Candidato de reducibilidad)

Un conjunto de términos $\mathscr{C} \subseteq \Lambda$ (posiblemente abiertos) es un candidato de reducibilidad cuando cumple los siguientes criterios:

(CR1) $\mathscr{C} \subseteq \mathbf{SN}$

(CR2) Si $M \in \mathscr{C}$, entonces $\mathrm{Red}_1(M) \subseteq \mathscr{C}$

(CR3) Si un término neutro M es tal que $\mathrm{Red}_1(M) \subseteq \mathscr{C}$, entonces $M \in \mathscr{C}$

Obs.: Por (CR3), un candidato de reducibilidad contiene todas las variables

- Se escribe $\mathcal{CR}$ al conjunto de todos los candidatos de reducibilidad

Propiedades de clausura por $\beta\delta$-expansión

Lema (Clausura por $\beta\delta$-expansión)

Dado un candidato de reducibilidad $\mathscr{C}$:

1. Si $M[x := N] \in \mathscr{C}$ y $A, N \in \mathbf{SN}$, entonces $(\lambda x : A \,.\, M)N \in \mathscr{C}$
2. Si $p \in \mathscr{C}$ y $P \in \mathbf{SN}$, entonces $\mathtt{unit\_rec}\ P\ p\ \langle\rangle \in \mathscr{C}$
3. Si $p \in \mathscr{C}$ y $P, q \in \mathbf{SN}$, entonces $\mathtt{bool\_rec}\ P\ p\ q\ \mathtt{true} \in \mathscr{C}$
4. Si $q \in \mathscr{C}$ y $P, p \in \mathbf{SN}$, entonces $\mathtt{bool\_rec}\ P\ p\ q\ \mathtt{false} \in \mathscr{C}$
5. Si $p \in \mathscr{C}$ y $P, f \in \mathbf{SN}$, entonces $\mathtt{nat\_rec}\ P\ p\ f\ 0 \in \mathscr{C}$
6. Si $f\ n\ (\mathtt{nat\_rec}\ P\ p\ f\ n) \in \mathscr{C}$, entonces $\mathtt{nat\_rec}\ P\ p\ f\ (\mathtt{S}\, n) \in \mathscr{C}$
7. Si $P \in \mathscr{C}$ y $F \in \mathbf{SN}$, entonces $\mathtt{nat\_rect}\ P\ F\ 0 \in \mathscr{C}$
8. Si $f\ n\ (\mathtt{nat\_rect}\ P\ F\ n) \in \mathscr{C}$, entonces $\mathtt{nat\_rect}\ P\ f\ (\mathtt{S}\, n) \in \mathscr{C}$
9. Si $f\ a\ b \in \mathscr{C}$ y $A, B \in \mathbf{SN}$, entonces $\mathtt{prod\_rec}\ A\ B\ P\ f\ (\mathtt{pair}\ A\ B\ a\ b) \in \mathscr{C}$
10. Si $f\ a \in \mathscr{C}$ y $A, B, g \in \mathbf{SN}$, entonces $\mathtt{sum\_rec}\ A\ B\ P\ f\ g\ (\mathtt{inl}\ A\ B\ a) \in \mathscr{C}$
11. Si $g\ b \in \mathscr{C}$ y $A, B, f \in \mathbf{SN}$, entonces $\mathtt{sum\_rec}\ A\ B\ P\ f\ g\ (\mathtt{inr}\ A\ B\ b) \in \mathscr{C}$
12. Si $f\ a\ b \in \mathscr{C}$ y $A, B \in \mathbf{SN}$, entonces $\mathtt{dsum\_rec}\ A\ B\ P\ f\ (\mathtt{dpair}\ A\ B\ a\ b) \in \mathscr{C}$
13. Si $p \in \mathscr{C}$ y $A, P, q \in \mathbf{SN}$, entonces $\mathtt{eq\_elim}\ A\ P\ a\ a\ p\ (\mathtt{refl}\ A\ a) \in \mathscr{C}$

Demostración. Ejercicio.

Clausura por (CR3)

- Dado un conjunto de términos S, se define inductivamente el conjunto $\overline{S}$ («clausura de S por (CR3)») por las reglas:
  1. Si $M \in S$, entonces $M \in \overline{S}$
  2. Si M es neutro y $\mathrm{Red}_1(M) \subseteq \overline{S}$, entonces $M \in \overline{S}$
     (**Recordatorio:** $\mathrm{Red}_1(M) = \{M'_1, \ldots, M'_n\}$ siempre es finito)
- Por def., $\overline{S}$ es el mínimo superconjunto de S que cumple (CR3)
- **Ejemplo:** $\mathbf{HN} := \overline{\varnothing}$ (= conjunto de los términos hereditariamente neutros)

Proposición

Si S cumple (CR1) y (CR2), entonces $\overline{S}$ es un candidato de reducibilidad

$\Rightarrow$ $\overline{S}$ es el candidato de reducibilidad generado por S

- $\mathbf{HN}$ es el mínimo candidato de reducibilidad
- $\mathbf{SN}$ es el máximo candidato de reducibilidad

Construcción de candidatos (1/2)

- Candidatos asociados a los tipos básicos:
  - $\mathscr{C}_{\mathbf{0}} := \overline{\varnothing} \quad (= \mathbf{HN})$
  - $\mathscr{C}_{\mathbf{1}} := \overline{\{\langle\rangle\}}$
  - $\mathscr{C}_{\mathtt{Bool}} := \overline{\{\mathtt{true};\ \mathtt{false}\}}$
  - $\mathscr{C}_{\mathtt{Nat}} := \overline{\{\mathtt{S}^n\,\mathtt{0} \ :\ n \in \mathbb{N}\}}$
- Candidato asociado al tipo identidad:
  - Para todos $M, M' \in \mathbf{SN}$ se escribe:

$$\mathscr{C}_{\mathtt{Eq}(M,M')} := \begin{cases} \mathscr{C}_{\mathtt{refl}} & \text{si } M \cong M' \\ \mathbf{HN} & \text{si no} \end{cases}$$

con $\quad \mathscr{C}_{\mathtt{refl}} := \overline{\{\mathtt{refl}\,A\,M \ :\ A, M \in \mathbf{SN}\}}$

Obs.: No se necesita suponer más que $A, M \in \mathbf{SN}$ en la definición del candidato $\mathscr{C}_{\mathtt{refl}}$, pues A y M son computacionalmente irrelevantes en la construcción $\mathtt{refl}\,A\,M$ (sólo sirven para el tipado)

Construcción de candidatos (2/2)

- Producto y suma dependientes de una familia de candidatos:
  - Dados $\mathscr{C} \in \mathcal{CR}$ y $\mathscr{D}_N \in \mathcal{CR}$ para todo $N \in \mathscr{C}$, se definen:

$$\prod_{N \in \mathscr{C}} \mathscr{D}_N = \{M \in \Lambda \ : \ (\forall N \in \mathscr{C})\ M\,N \in \mathscr{D}_N\}$$

$$\sum_{N \in \mathscr{C}} \mathscr{D}_N = \overline{\{\mathtt{dpair}\,A\,B\,N\,M \ : \ A, B \in \mathbf{SN},\ N \in \mathscr{C},\ M \in \mathscr{D}_N\}}$$

Obs.: A y B son computacionalmente irrelevantes en la def. de $\sum_{N \in \mathscr{C}} \mathscr{D}_N$

- Producto y suma de candidatos:
  - $\mathscr{C} \times \mathscr{D} := \overline{\{\mathtt{pair}\,A\,B\,M\,N \ : \ A, B \in \mathbf{SN},\ M \in \mathscr{C},\ N \in \mathscr{D}\}}$
  - $\mathscr{C} + \mathscr{D} := \overline{\mathtt{inl}(\mathscr{C}) \cup \mathtt{inr}(\mathscr{D})}$

con $\mathtt{inl}(\mathscr{C}) := \{\mathtt{inl}\,A\,B\,M \ : \ A, B \in \mathbf{SN},\ M \in \mathscr{C}\}$
$\mathtt{inr}(\mathscr{D}) := \{\mathtt{inr}\,A\,B\,M \ : \ A, B \in \mathbf{SN},\ M \in \mathscr{D}\}$

Obs.: Misma observación sobre A y B en las def. de $\mathscr{C} \times \mathscr{D}$ y $\mathscr{C} + \mathscr{D}$

Interpretación de los tipos pequeños (1/3)

Se define por inducción simultánea:

- Un conjunto de términos $\Phi_0 \subseteq \Lambda$ (interpretación de Set)
- Una función $\phi_0 : \Phi_0 \to \mathcal{CR}$ (interpretación de los tipos pequeños)

Definición inductiva de Φ_0 y $\phi_0 : \Phi_0 \to \mathcal{CR}$ (cláusulas 1–6/10):

(1) $\mathbf{0} \in \Phi_0$ y $\phi_0(\mathbf{0}) := \mathscr{C}_{\mathbf{0}}$

(2) $\mathbf{1} \in \Phi_0$ y $\phi_0(\mathbf{1}) := \mathscr{C}_{\mathbf{1}}$

(3) $\mathtt{Bool} \in \Phi_0$ y $\phi_0(\mathtt{Bool}) := \mathscr{C}_{\mathtt{Bool}}$

(4) $\mathtt{Nat} \in \Phi_0$ y $\phi_0(\mathtt{Nat}) := \mathscr{C}_{\mathtt{Nat}}$

(5) Si $A \in \Phi_0$ y $M, M' \in \phi_0(A)$,
entonces $(\mathtt{Eq}\, A\, M\, M') \in \Phi_0$ y $\phi_0(\mathtt{Eq}\, A\, M\, M') := \mathscr{C}_{\mathtt{Eq}(M,M')}$

(6) Si $A \in \Phi_0$ y $B[x := N] \in \Phi_0$ para todo $N \in \phi_0(A)$,
entonces $(\Pi x : A\,.\,B) \in \Phi_0$ y

$$\phi_0(\Pi x : A\,.\,B) := \prod_{N \in \phi_0(A)} \phi_0(B[x := N]) \qquad (\ldots)$$

Interpretación de los tipos pequeños (2/3)

Definición inductiva de Φ_0 y $\phi_0 : \Phi_0 \to \mathcal{CR}$ (cláusulas 7–10/10):

(7) Si $A \in \Phi_0$ y $B \in \Phi_0$,
entonces $(A \times B) \in \Phi_0$ y $\phi_0(A \times B) := \phi_0(A) \times \phi_0(B)$

(8) Si $A \in \Phi_0$ y $B \in \Phi_0$,
entonces $(A + B) \in \Phi_0$ y $\phi_0(A + B) := \phi_0(A) + \phi_0(B)$

(9) Si $A \in \Phi_0$ y $B\,N \in \Phi_0$ para todo $N \in \phi_0(A)$,
entonces $(\mathtt{DSum}\,A\,B) \in \Phi_0$ y

$$\phi_0(\mathtt{DSum}\,A\,B) := \sum_{N \in \phi_0(A)} \phi_0(B\,N)$$

(10) Si A es un término neutro tal que $A' \in \Phi_0$ para todo $A' \in \mathsf{Red}_1(A)$,
entonces $A \in \Phi_0$ y

$$\phi_0(A) := \overline{\left(\bigcup_{A' \in \mathsf{Red}_1(A)} \phi_0(A') \right)}$$

Interpretación de los tipos pequeños (3/3)

- Los 10 casos de la definición inductiva de Φ_0 son disjuntos
  $\Rightarrow$ el árbol de derivación de cada $A \in \Phi_0$ es único
  $\Rightarrow$ la función $\phi_0 : \Phi_0 \to \mathcal{CR}$ está bien definida

Lema (Invariancia por reducción)

1. Φ_0 es un candidato de reducibilidad
2. Para todo $A \in \Phi_0$, tenemos que $\phi_0(A) = \phi_0(\downarrow A)$

Demostración.

1. (CR1) y (CR2) se demuestran por inducción sobre la derivación de $A \in \Phi_0$; (CR3) sigue de la cláusula (10) de la definición de Φ_0.
2. Por induccón sobre la derivación de $A \in \Phi_0$. □

- **Obs.:** El lema permite observar que cuando A es un término neutro tal que $A' \in \Phi_0$ para todo $A' \in \mathrm{Red}_1(A)$ (cláusula (10)), se tiene que:

$$\phi_0(A) = \begin{cases} \mathbf{HN} & \text{si } \mathrm{Red}_1(A) = \varnothing \\ \phi_0(A') & \text{para cualquier } A' \in \mathrm{Red}_1(A) \text{ si no} \end{cases}$$

Interpretación de los tipos grandes (y pequeños)

- Se define (de vuelta) por inducción simultánea:
  - Un conjunto de términos $\Phi \subseteq \Lambda$ (interpretación de Type [3])
  - Una función $\phi : \Phi \to \mathcal{CR}$ (interpretación de los tipos grandes)
- Definición inductiva de Φ y $\phi : \Phi \to \mathcal{CR}$:
  - Se reutilizan las cláusulas (1)–(10) que definen Φ_0 y $\phi_0 : \Phi_0 \to \mathcal{CR}$ (remplazando en cada cláusula Φ_0 por Φ y ϕ_0 por ϕ)
  - Se añade la cláusula: (11) $\mathsf{Set} \in \Phi$ y $\phi(\mathsf{Set}) := \Phi_0$
- **Obs.:** Por construcción, tenemos que $\Phi_0 \subset \Phi$ y $\phi_0 = \phi_{|\Phi_0}$

Lema (Invariancia por reducción)

1. Φ es un candidato de reducibilidad
2. Para todo $A \in \Phi$, tenemos que $\phi(A) = \phi(\downarrow A)$

Demostración. Análoga a la demostración del lema anterior. □

[3] En la construcción de Φ y ϕ, se supone implícitamente que $\mathsf{Set} \subset \mathsf{Type}$

Interpretación de los contextos y normalización (1/2)

- Se asocia a cada contexto Γ un conjunto de sustituciones $[\![\Gamma]\!]$, definido por inducción sobre la lista Γ por:

$$\begin{array}{rcl} [\![\emptyset]\!] & := & \{\varnothing\} \qquad \text{(sustitución vacía)} \\ [\![\Gamma, x : A]\!] & := & \{\sigma \cup \{x := N\} \ : \ \sigma \in [\![\Gamma]\!], \ \ A[\sigma] \in \Phi \ \text{ y } \ N \in \phi(A[\sigma])\} \end{array}$$

- **¡Cuidado!** La definición se aplica a cualquier contexto sintáctico, incluso a los contextos mal formados. En muchos casos, la condición $A[\sigma] \in \Phi$ no se cumple para ninguna sustitución (pues A no es un tipo), de tal modo que $[\![\Gamma]\!] = \varnothing$.

Proposición (1er invariante de normalización)

Si $\Gamma \vdash M : A$, entonces para todo $\sigma \in [\![\Gamma]\!]$, tenemos que:

$$A[\sigma] \ \in \ \Phi \cup \{\mathsf{Type}\} \qquad \text{y} \qquad M[\sigma] \ \in \ \begin{cases} \Phi & \text{si } A \equiv \mathsf{Type} \\ \phi(A[\sigma]) & \text{si no} \end{cases}$$

Demostración. Por inducción sobre la derivación de $\Gamma \vdash M : T$. □

Ejercicio. Detallar la prueba.

Interpretación de los contextos y normalización (2/2)

- Dado un contexto $\Gamma \equiv x_1 : A_1, \ldots, x_n : A_n$, se escribe

$$id_\Gamma :\equiv [x_1 := x_1; \ldots; x_n := x_n] \qquad \text{(sustitución identidad)}$$

Proposición (2^{do} invariante de normalización)

Si $\vdash \Gamma$ context, entonces $id_\Gamma \in [\![\Gamma]\!]$

Demostración. Por inducción sobre la derivación de $\vdash \Gamma$ context, usando el 1^{er} invariante de normalización en cada etapa. □

Ejercicio. Detallar la prueba.

Corolario (Normalización fuerte)

Si $\Gamma \vdash M : A$, entonces M es fuertemente normalizante

Demostración. Basta con aplicar el 1^{er} invariante de normalización con la sustitución $\sigma :\equiv id_\Gamma \in [\![\Gamma]\!]$ (por el 2^{do} invariante de normalización). □

Observaciones

- El núcleo de la prueba es la definición por inducción simultánea del conjunto Φ y de la función $\phi : \Phi \to \mathcal{CR}$.

  (Definición en dos etapas, para interpretar Set y luego Type)

- Esta construcción está considerada como predicativa, y se puede formalizar en algunas extensiones de la teoría de tipos de Martin-Löf con un mecanismo de definición inductiva-recursiva [Dybjer 2000]

- Cabe destacar que la misma construcción se puede formalizar mediante una construcción impredicativa en HA2 (¡Ejercicio!)
  Y por lo tanto:

$$\text{MLTT} < \text{HA2}$$

- Hasta ahora, y a pesar de su considerable expresividad, todas las versiones de la teoría de tipos de Martin-Löf tienen una fuerza teórica (estrictamente) menor que la de HA2

Plan

Otros tipos inductivos: las listas (polimórficas)

$$\begin{array}{lcl}
\texttt{List} & : & \mathsf{Set} \to \mathsf{Set} \\
\\
\texttt{nil} & : & \Pi A : \mathsf{Set}\,.\ \texttt{List}\ A \\
\texttt{cons} & : & \Pi A : \mathsf{Set}\,.\ A \to \texttt{List}\ A \to \texttt{List}\ A \\
\\
\texttt{list_rec} & : & \Pi A : \mathsf{Set}\,.\ \Pi X : \texttt{List}\ A \to \mathsf{Set}\,. \\
 & & \quad X\ (\texttt{nil}\ A) \to \\
 & & \quad\quad (\Pi x : A\,.\,\Pi l : \texttt{List}\ A\,.\ X\ l \to X\ (\texttt{cons}\ A\ x\ l)) \to \\
 & & \quad\quad\quad \Pi l : \texttt{List}\ A\,.\,X\ l
\end{array}$$

δ-reglas asociadas

$$\begin{array}{lcl}
\texttt{list_rec}\ A\ X\ x\ f\ (\texttt{nil}\ A) & \succ & x \\
\texttt{list_rec}\ A\ X\ x\ f\ (\texttt{cons}\ A\ a\ l) & \succ & f\ a\ l\ (\texttt{list_rec}\ A\ X\ x\ f\ l)
\end{array}$$

Ejercicio: Implementar las funciones:

- $\texttt{length} : \Pi A : \mathsf{Set}\,.\,\texttt{List}\ A \to \texttt{Nat}$
- $\texttt{concat} : \Pi A : \mathsf{Set}\,.\,\texttt{List}\ A \to \texttt{List}\ A \to \texttt{List}\ A$

Otros tipos inductivos: las listas dependientes («vectores»)

$$
\begin{array}{lcl}
\mathtt{Vect} & : & \mathsf{Set} \to \mathtt{Nat} \to \mathsf{Set} \\
\\
\mathtt{vnil} & : & \Pi A : \mathsf{Set} \,.\; \mathtt{Vect}\; A\; \mathtt{0} \\
\mathtt{vcons} & : & \Pi A : \mathsf{Set} \,.\; \Pi n : \mathtt{Nat} \,.\; A \to \mathtt{Vect}\; A\; n \to \mathtt{Vect}\; A\; (\mathtt{S}\; n) \\
\\
\mathtt{vect\_rec} & : & \Pi A : \mathsf{Set} \,.\; \Pi X : (\Pi n : \mathtt{Nat} \,.\, \mathtt{Vect}\; A\; n \to \mathsf{Set}) \,. \\
 & & \quad X\; \mathtt{0}\; (\mathtt{vnil}\; A) \to \\
 & & \quad\quad (\Pi n : \mathtt{Nat} \,.\, \Pi x : A \,.\, \Pi v : \mathtt{Vect}\; A\; n \,. \\
 & & \quad\quad\quad X\; n\; v \to X\; (\mathtt{S}\; n)\; (\mathtt{vcons}\; A\; n\; x\; v)) \to \\
 & & \quad\quad\quad \Pi n : \mathtt{Nat} \,.\; \Pi v : \mathtt{Vect}\; A\; n \,.\; X\; n\; v
\end{array}
$$

δ-reglas asociadas

$$
\begin{array}{lcl}
\mathtt{vect\_rec}\; A\; X\; x\; f\; \mathtt{0}\; (\mathtt{vnil}\; A) & \succ & x \\
\mathtt{vect\_rec}\; A\; X\; x\; f\; (\mathtt{S}\; n)\; (\mathtt{vcons}\; A\; n\; a\; v) & \succ & f\; n\; a\; v\; (\mathtt{vect\_rec}\; A\; X\; x\; f\; n\; v)
\end{array}
$$

Ejercicio: Implementar las funciones:

- $\mathtt{vlength} : \Pi A : \mathsf{Set} \,.\, \Pi n : \mathtt{Nat} \,.\, \mathtt{Vect}\; A\; n \to \mathtt{Nat}$ (¡2 soluciones!)
- $\mathtt{vconcat} : \Pi A : \mathsf{Set} \,.\, \Pi n : \mathtt{Nat} \,.\, \mathtt{Vect}\; A\; n \to \Pi m : \mathtt{Nat} \,.\, \mathtt{Vect}\; A\; m \to \mathtt{Vect}\; A\; (\mathtt{plus}\; n\; m)$

Otros tipos inductivos: los ordinales numerables

$$
\begin{array}{lcl}
\mathtt{Ord} & : & \mathsf{Set} \\
\\
\mathtt{0'} & : & \mathtt{Ord} \\
\mathtt{S'} & : & \mathtt{Ord} \to \mathtt{Ord} \\
\mathtt{lim} & : & (\mathtt{Nat} \to \mathtt{Ord}) \to \mathtt{Ord} \\
\\
\mathtt{ord\_rec} & : & \Pi X : \mathtt{Ord} \to \mathsf{Set}\,. \\
 & & \quad X\,\mathtt{0'} \;\to\; (\Pi x : \mathtt{Ord}\,.\, X\,x \to X(\mathtt{S'}\,x)) \;\to \\
 & & \quad\quad (\Pi f : \mathtt{Nat} \to \mathtt{Ord}\,.\, (\Pi n : \mathtt{Nat}\,.\, X\,(f\,n)) \to X\,(\mathtt{lim}\,f)) \;\to \\
 & & \quad\quad\quad \Pi x : \mathtt{Ord}\,.\;\; X\,x
\end{array}
$$

δ-reglas asociadas

$$
\begin{array}{lcl}
\mathtt{ord\_rec}\,X\,x\,g\,h\,\mathtt{0'} & \succ & x \\
\mathtt{ord\_rec}\,X\,x\,g\,h\,(\mathtt{S'}\,z) & \succ & g\,z\,(\mathtt{ord\_rec}\,X\,x\,g\,h\,z) \\
\mathtt{ord\_rec}\,X\,x\,g\,h\,(\mathtt{lim}\,f) & \succ & h\,f\,(\lambda n : \mathtt{Nat}\,.\,\mathtt{ord\_rec}\,X\,x\,g\,h\,(f\,n))
\end{array}
$$

Obs.: Observar (en la 3$^{\text{ra}}$ regla) la llamada recursiva bajo la λ

La jerarquía de universos predicativos

- Se puede añadir una jerarquía infinita de universos predicativos:

$$\mathsf{Type}_0(=\mathsf{Set}) \in \mathsf{Type}_1(=\mathsf{Type}) \in \mathsf{Type}_2 \in \mathsf{Type}_3 \in \cdots$$

- Modificación de las reglas de tipado:

$$\frac{\Gamma \vdash A : \mathsf{Type}_i}{\vdash \Gamma, x : A \text{ context}}\ x \notin \mathsf{dom}(\Gamma) \qquad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash A' : \mathsf{Type}_i}{\Gamma \vdash M : A'}\ A \cong A'$$

$$\frac{\vdash \Gamma \text{ context}}{\Gamma \vdash \mathsf{Type}_i : \mathsf{Type}_{i+1}} \qquad \frac{\Gamma \vdash A : \mathsf{Type}_i \qquad \Gamma, x : A \vdash B : \mathsf{Type}_j}{\Gamma \vdash \Pi x : A \,.\, B : \mathsf{Type}_{\max(i,j)}}$$

- Los tipos inductivos están definidos en todos los universos
- También se puede añadir una regla de cumulatividad:

$$\frac{\Gamma \vdash M : \mathsf{Type}_i}{\Gamma \vdash M : \mathsf{Type}_{i+1}}$$

(¡Se pierde la unicidad del tipo!)

Ejercicio: Adaptar el formalismo (tipos inductivos) + prueba de normalización

¿Unicidad de las pruebas de identidad?

- El tipo identidad $\mathtt{Eq}\,A\,x\,y$ (escrito $x =_A y$, con $A : \mathsf{Set}$, $x, y : A$) está definido a partir de un único constructor

$$\mathtt{refl} \quad : \quad \forall A : \mathsf{Set}\,.\,\forall x : A\,.\;\mathtt{Eq}\,A\,x\,x$$

- Sin embargo, no se puede demostrar en MLTT que las pruebas de igualdad son únicas:

$$\mathtt{MLTT} \quad \not\vdash \quad \forall A : \mathsf{Set}\,.\,\forall x, y : A\,.\,\forall e, e' : \mathtt{Eq}\,A\,x\,y\,.\;\mathtt{Eq}\,(\mathtt{Eq}\,A\,x\,y)\,e\,e' \qquad (1)$$

$$\not\vdash \quad \forall A : \mathsf{Set}\,.\,\forall x : A\,.\,\forall e : \mathtt{Eq}\,A\,x\,x\,.\;\mathtt{Eq}\,(\mathtt{Eq}\,A\,x\,x)\,e\,(\mathtt{refl}\,A\,x) \qquad (2)$$

  Ejercicio: Demostrar que (1) y (2) son equivalentes en MLTT

- En efecto (1) y (2) son independientes de MLTT:
  - Se puede extender MLTT (de modo consistente) con el axioma **K** de Streicher, de tipo (2) y con la δ-regla adecuada – Ejercicio
  - Por otro lado, el modelo de los grupoides [Hofmann & Streicher 2002] refuta la unicidad de las pruebas de identidad

- ¿Cuál es la opción más interesante?

Teoría de tipos homotópicos (1/2)

Entre 2006 y 2009, Voyevodski[4] trabajó sobre una interpretación «topológica» de MLTT, en la cual:

- Cada tipo A es un espacio topológico
- Cada prueba $e : \mathtt{Eq}\, A\, x\, y$ es un camino de x a y en A
- Según esta interpretación:

$$\mathtt{eq\_refl}\, A\, x \quad : \quad \mathtt{Eq}\, A\, x\, x$$

$\rightsquigarrow$ camino identidad

$$\mathtt{eq\_sym}\, A\, x\, y \quad : \quad \mathtt{Eq}\, A\, x\, y \to \mathtt{Eq}\, A\, y\, x$$

$\rightsquigarrow$ operador de construcción del camino simétrico

$$\mathtt{eq\_trans}\, A\, x\, y\, z \quad : \quad \mathtt{Eq}\, A\, x\, y \to \mathtt{Eq}\, A\, y\, z \to \mathtt{Eq}\, A\, x\, z$$

$\rightsquigarrow$ operador de composición de caminos

- En particular, una prueba de identidad $h : \mathtt{Eq}\,(\mathtt{Eq}\, A\, x\, y)\, e\, e'$ entre dos pruebas $e, e' : \mathtt{Eq}\, A\, x\, y$ es una homotopía $h : e \Rightarrow e'$

[4] Vladímir Voyevodski (1966–2017), ganador de la medalla Fields en 2002

Teoría de tipos homotópicos (2/2)

A partir de esas ideas surgió la teoría de tipos homotópicos (HoTT)

$$\text{HoTT} \quad = \quad \text{MLTT} + \text{axioma de univalencia}$$

- Dos tipos $A, B : \mathsf{Type}_i$ son isomorfos (notación $A \sim B$) cuando existen funciones $f : A \to B$ y $g : B \to A$ tales que

$$g \circ f \;=_{A \to A}\; id_A \qquad \text{y} \qquad f \circ g \;=_{B \to B}\; id_B$$

- Intuitivamente, las pruebas

$$e \;:\; \mathtt{Eq}\,(A \to A)\,(g \circ f)\,id_A \qquad \text{y} \qquad e' \;:\; \mathtt{Eq}\,(B \to B)\,(f \circ g)\,id_B$$

  definen una equivalencia de homotopía entre los espacios A y B

- Obviamente: $A =_{\mathsf{Type}_i} B \;\to\; A \sim B.$ Esto sugiere el:

Axioma de univalencia

$$\forall A, B : \mathsf{Type}_i\,.\;\; A \sim B \;\to\; A =_{\mathsf{Type}_i} B$$

- Contenido computacional del axioma? Modelos? Aplicaciones?