

Teoría de categorías y cálculo lambda

Mario Román García

24 de junio de 2018

Grado en Ingeniería Informática y Matemáticas - Universidad de Granada

- 1. Cálculo lambda no tipado
- 2. Cálculo lambda tipado
- 3. Categorías cartesianas
- 4. Categorías localmente cartesianas
- 5. Teoría de tipos
- 6. Matemática constructivista

La idea de este trabajo es relacionar las fundaciones de la programación y las matemáticas.

1. Cálculo lambda no tipado

2. Cálculo lambda tipado

3. Categorías cartesianas

4. Categorías localmente cartesianas

5. Teoría de tipos

6. Matemática constructivista

El **cálculo lambda** es un sistema formal dado por ecuaciones sobre términos lambda. Abstracción y aplicación son nociones primitivas.

$$\text{Expr} := \left\{ \begin{array}{ll} x, & \text{variables, de un conjunto numerable,} \\ \text{Expr Expr}, & \text{aplicación de funciones,} \\ \lambda x. \text{Expr}, & \text{abstracción sobre una variable.} \end{array} \right.$$

Consideramos

- α -equivalencia, invariancia a renombramientos $(\lambda x.M[x]) \equiv (\lambda y.M[y])$;
- β -reducciones, aplicación de funciones $(\lambda x.M) N \longrightarrow_{\beta} M_{[N/x]}$;
- η -reducciones, extensionalidad $(\lambda x.f \ x) \equiv f$.

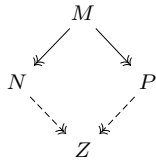
1. El cálculo lambda es un lenguaje y un sistema formal que captura las nociones de abstracción y aplicación.
2. Se le aplican varias reglas:
3. Las variables son intercambiables.
4. La aplicación es mediante sustitución.
5. La abstracción y aplicación son duales.

Alonzo Church. “A set of postulates for the foundation of logic”. En: Annals of mathematics (1932), págs. 346-366.

H.P. Barendregt. The lambda calculus: its syntax and semantics. Studies in logic and the foundations of mathematics. North-Holland, 1984. ISBN: 9780444867483.

Buscamos **formas normales** invariantes a reducciones. Existen términos que divergen como $\Omega = (\lambda x.(xx))(\lambda x.(xx))$.

Teorema (Church-Rosser)
La reducción es confluente.



Si existe la forma normal, es única.

Hay expresiones que se reducirán o no dependiendo del orden de evaluación, como $(\lambda x.\lambda y.y) \Omega (\lambda x.x)$.

Teorema (Barendregt)
Si existe una forma normal del término lambda, la estrategia que reduce a cada paso la aplicación más a la izquierda la encuentra.

La reducción da una forma de cálculo **Turing-completa**.

1. Estas propiedades hacen del cálculo lambda un lenguaje de programación útil.
2. No todas las computaciones terminan, pero cuando lo hacen llegan a forma normal, esto es Church-Rosser.
3. Podríamos reducirlas de muchas formas, pero tenemos una estrategia que funciona siempre que es posible.

Robert Pollack. “Polishing Up the Tait-Martin-Löf Proof of the Church-Rosser Theorem”. En: Proc. De Winternöte, Chalmers University (1995).

Ryo Kashima. “A Proof of the Standardization Theorem in Lambda-Calculus”. En: Tokyo Institute of Technology (2000).

mario@kosmos ~ mikrokosmos

Welcome to the Mikrokosmos Lambda Interpreter!

Version 0.7.0. GNU General Public License Version 3.

mikro>

1. Así que vamos a implementarlo como un lenguaje de programación.
2. Mikrokosmos es un intérprete escrito en Haskell, que facilita tratar expresiones simbólicas.
3. Usamos una técnica de Church para escribir definiciones de Peano inductivas.
4. Y así podemos escribir definiciones inductivas, como la suma.
5. Podemos utilizar estructuras de datos más complejas, como listas.
6. E incluso, estructuras infinitas. Aquí aprovecho el teorema que dice que la reducción siempre termina.
7. Como ejemplo, el operador μ de Gödel, que encuentra el primer natural que cumple una condición.

Welcome to the Mikrokosmos Lambda Interpreter!

Version 0.7.0. GNU General Public License Version 3.

```
mikro> 2 = \s.\z.s (s z)
```

```
mikro> 
```

1. Así que vamos a implementarlo como un lenguaje de programación.
2. Mikrokosmos es un intérprete escrito en Haskell, que facilita tratar expresiones simbólicas.
3. Usamos una técnica de Church para escribir definiciones de Peano inductivas.
4. Y así podemos escribir definiciones inductivas, como la suma.
5. Podemos utilizar estructuras de datos más complejas, como listas.
6. E incluso, estructuras infinitas. Aquí aprovecho el teorema que dice que la reducción siempre termina.
7. Como ejemplo, el operador mu de Gödel, que encuentra el primer natural que cumple una condición.

Welcome to the Mikrokosmos Lambda Interpreter!

Version 0.7.0. GNU General Public License Version 3.

```
mikro> 2 = \s.\z.s (s z)
mikro> plus = \n.\m.n succ m
mikro> plus 3 4
λa.λb.a (a (a (a (a (a (a b)))))) ⇒ 7
mikro> 
```

1. Así que vamos a implementarlo como un lenguaje de programación.
2. Mikrokosmos es un intérprete escrito en Haskell, que facilita tratar expresiones simbólicas.
3. Usamos una técnica de Church para escribir definiciones de Peano inductivas.
4. Y así podemos escribir definiciones inductivas, como la suma.
5. Podemos utilizar estructuras de datos más complejas, como listas.
6. E incluso, estructuras infinitas. Aquí aprovecho el teorema que dice que la reducción siempre termina.
7. Como ejemplo, el operador mu de Gödel, que encuentra el primer natural que cumple una condición.

Welcome to the Mikrokosmos Lambda Interpreter!

Version 0.7.0. GNU General Public License Version 3.

```
mikro> 2 = \s.\z.s (s z)
mikro> plus = \n.\m.n succ m
mikro> plus 3 4
λa.λb.a (a (a (a (a (a (a b)))))) ⇒ 7
mikro> sum = fold plus 0
mikro> sum (cons 1 (cons 2 (cons 3 nil)))
λa.λb.a (a (a (a (a (a b)))))) ⇒ 6
mikro> 
```

1. Así que vamos a implementarlo como un lenguaje de programación.
2. Mikrokosmos es un intérprete escrito en Haskell, que facilita tratar expresiones simbólicas.
3. Usamos una técnica de Church para escribir definiciones de Peano inductivas.
4. Y así podemos escribir definiciones inductivas, como la suma.
5. Podemos utilizar estructuras de datos más complejas, como listas.
6. E incluso, estructuras infinitas. Aquí aprovecho el teorema que dice que la reducción siempre termina.
7. Como ejemplo, el operador mu de Gödel, que encuentra el primer natural que cumple una condición.

Welcome to the Mikrokosmos Lambda Interpreter!

Version 0.7.0. GNU General Public License Version 3.

```
mikro> 2 = \s.\z.s (s z)
mikro> plus = \n.\m.n succ m
mikro> plus 3 4
λa.λb.a (a (a (a (a (a (a b)))))) ⇒ 7
mikro> sum = fold plus 0
mikro> sum (cons 1 (cons 2 (cons 3 nil)))
λa.λb.a (a (a (a (a (a b))))) ⇒ 6
mikro> naturals := fix (compose (cons 0) (map (plus 1)))
mikro> sum (take 5 naturals)
λa.λb.a (a (a (a (a (a (a (a (a (a (a b)))))))))) ⇒ 10
mikro> □
```

1. Así que vamos a implementarlo como un lenguaje de programación.
2. Mikrokosmos es un intérprete escrito en Haskell, que facilita tratar expresiones simbólicas.
3. Usamos una técnica de Church para escribir definiciones de Peano inductivas.
4. Y así podemos escribir definiciones inductivas, como la suma.
5. Podemos utilizar estructuras de datos más complejas, como listas.
6. E incluso, estructuras infinitas. Aquí aprovecho el teorema que dice que la reducción siempre termina.
7. Como ejemplo, el operador mu de Gödel, que encuentra el primer natural que cumple una condición.

Welcome to the Mikrokosmos Lambda Interpreter!

Version 0.7.0. GNU General Public License Version 3.

```
mikro> 2 = \s.\z.s (s z)
mikro> plus = \n.\m.n succ m
mikro> plus 3 4
λa.λb.a (a (a (a (a (a (a b)))))) ⇒ 7
mikro> sum = fold plus 0
mikro> sum (cons 1 (cons 2 (cons 3 nil)))
λa.λb.a (a (a (a (a (a b))))) ⇒ 6
mikro> naturals := fix (compose (cons 0) (map (plus 1)))
mikro> sum (take 5 naturals)
λa.λb.a (a (a (a (a (a (a (a (a (a (a b)))))))))) ⇒ 10
mikro> mu (\n.eq (mult 3 n) (plus 6 n))
λa.λb.a (a (a b)) ⇒ 3
mikro> □
```

1. Así que vamos a implementarlo como un lenguaje de programación.
2. Mikrokosmos es un intérprete escrito en Haskell, que facilita tratar expresiones simbólicas.
3. Usamos una técnica de Church para escribir definiciones de Peano inductivas.
4. Y así podemos escribir definiciones inductivas, como la suma.
5. Podemos utilizar estructuras de datos más complejas, como listas.
6. E incluso, estructuras infinitas. Aquí aprovecho el teorema que dice que la reducción siempre termina.
7. Como ejemplo, el operador mu de Gödel, que encuentra el primer natural que cumple una condición.

1. Cálculo lambda no tipado

2. Cálculo lambda tipado

3. Categorías cartesianas

4. Categorías localmente cartesianas

5. Teoría de tipos

6. Matemática constructivista

El cálculo lambda simplemente tipado consta de tres constructores de tipos: un tipo **unidad** con un solo elemento 1, un **producto** para cada dos tipos, y el tipo **función** entre dos tipos cualquiera.

$$\frac{}{\Gamma \vdash * : 1} \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \quad \frac{\Gamma \vdash m : A \times B}{\Gamma \vdash \pi_1 m : A} \quad \frac{\Gamma \vdash m : A \times B}{\Gamma \vdash \pi_2 m : B}$$
$$\frac{\Gamma, x : A \vdash m : B}{\Gamma \vdash \lambda x. m : A \rightarrow B} \quad \frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f a : B}$$

- 1. Ahora añadimos tipos al cálculo lambda.
- 2. Tomamos unas reglas de tipado, por ejemplo, dados a y b, creamos el par (a,b).
- 3. Tenemos tipos producto, tipos función, tipos unión, y tipos vacíos.
- 4. No todo término vamos a poder tiparlo, pero los tipados normalizan siempre.

Jean-Yves Girard, Paul Taylor e Yves Lafont. Proofs and Types. New York, NY, USA: Cambridge University Press, 1989. ISBN: 0-521-37181-3.

El cálculo lambda simplemente tipado consta de tres constructores de tipos: un tipo **unidad** con un solo elemento 1, un **producto** para cada dos tipos, y el tipo **función** entre dos tipos cualquiera. Además, podemos añadir un tipo **vacío** 0 y un tipo **unión** $A + B$.

$$\begin{array}{c} \frac{}{\Gamma \vdash * : 1} \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \quad \frac{\Gamma \vdash m : A \times B}{\Gamma \vdash \pi_1 m : A} \quad \frac{\Gamma \vdash m : A \times B}{\Gamma \vdash \pi_2 m : B} \\[10pt] \frac{\Gamma, x : A \vdash m : B}{\Gamma \vdash \lambda x. m : A \rightarrow B} \quad \frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f a : B} \\[10pt] \frac{\Gamma \vdash m : 0}{\Gamma \vdash \text{abort}_A m : A} \quad \frac{\Gamma \vdash a : A}{\Gamma \vdash \text{inl } a : A + B} \quad \frac{\Gamma \vdash b : B}{\Gamma \vdash \text{inr } b : A + B} \\[10pt] \frac{\Gamma \vdash m : A + B \quad \Gamma, a : A \vdash n : C \quad \Gamma, b : B \vdash p : C}{\Gamma \vdash (\text{case } m \text{ of } [a].n; [b].p) : C} \end{array}$$

1. Ahora añadimos tipos al cálculo lambda.
2. Tomamos unas reglas de tipado, por ejemplo, dados a y b, creamos el par (a,b).
3. Tenemos tipos producto, tipos función, tipos unión, y tipos vacíos.
4. No todo término vamos a poder tiparlo, pero los tipados normalizan siempre.

El cálculo lambda simplemente tipado consta de tres constructores de tipos: un tipo **unidad** con un solo elemento 1, un **producto** para cada dos tipos, y el tipo **función** entre dos tipos cualquiera. Además, podemos añadir un tipo **vacío** 0 y un tipo **unión** $A + B$.

$$\frac{}{\Gamma \vdash * : 1} \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \quad \frac{\Gamma \vdash m : A \times B}{\Gamma \vdash \pi_1 m : A} \quad \frac{\Gamma \vdash m : A \times B}{\Gamma \vdash \pi_2 m : B}$$
$$\frac{\Gamma, x : A \vdash m : B}{\Gamma \vdash \lambda x. m : A \rightarrow B} \quad \frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f a : B}$$
$$\frac{\Gamma \vdash m : 0}{\Gamma \vdash \text{abort}_A m : A} \quad \frac{\Gamma \vdash a : A}{\Gamma \vdash \text{inl } a : A + B} \quad \frac{\Gamma \vdash b : B}{\Gamma \vdash \text{inr } b : A + B}$$
$$\frac{\Gamma \vdash m : A + B \quad \Gamma, a : A \vdash n : C \quad \Gamma, b : B \vdash p : C}{\Gamma \vdash (\text{case } m \text{ of } [a].n; [b].p) : C}$$

Teorema (Tait, Girard)
Todo término tipado normaliza.

Jean-Yves Girard, Paul Taylor e Yves Lafont. Proofs and Types. New York, NY, USA: Cambridge University Press, 1989. ISBN: 0-521-37181-3.

1. Ahora añadimos tipos al cálculo lambda.
2. Tomamos unas reglas de tipado, por ejemplo, dados a y b, creamos el par (a,b).
3. Tenemos tipos producto, tipos función, tipos unión, y tipos vacíos.
4. No todo término vamos a poder tiparlo, pero los tipados normalizan siempre.

Heyting y Kolmogorov describen las implicaciones de la lógica intuicionista $A \rightarrow B$ como funciones que transforman demostraciones de A en transformaciones de B .

1. Resulta que en los años 30 Heyting y Kolmogorov propusieron una lectura de la lógica intuicionista tomando implicaciones como funciones.
2. La lógica intuicionista funciona igual que la clásica excepto porque no aceptamos el tercio excluso.
3. No es cierto que para toda A , se tenga (A o no A).

Heyting y Kolmogorov describen las implicaciones de la lógica intuicionista $A \rightarrow B$ como funciones que transforman demostraciones de A en transformaciones de B . Bajo esta interpretación el cálculo lambda es un sistema de demostraciones de la lógica proposicional intuicionista.

$$\frac{}{\Gamma \vdash * : 1} \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \quad \frac{\Gamma \vdash m : A \times B}{\Gamma \vdash \pi_1 m : A} \quad \frac{\Gamma \vdash m : A \times B}{\Gamma \vdash \pi_2 m : B}$$
$$\frac{\Gamma, x : A \vdash m : B}{\Gamma \vdash \lambda x. m : A \rightarrow B} \quad \frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f a : B}$$
$$\frac{\Gamma \vdash m : 0}{\Gamma \vdash \text{abort}_A m : A} \quad \frac{\Gamma \vdash a : A}{\Gamma \vdash \text{inl } a : A + B} \quad \frac{\Gamma \vdash b : B}{\Gamma \vdash \text{inr } b : A + B}$$
$$\frac{\Gamma \vdash m : A + B \quad \Gamma, a : A \vdash n : C \quad \Gamma, b : B \vdash p : C}{\Gamma \vdash (\text{case } m \text{ of } [a].n; [b].p) : C}$$

1. Resulta que en los años 30 Heyting y Kolmogorov propusieron una lectura de la lógica intuicionista tomando implicaciones como funciones.
2. La lógica intuicionista funciona igual que la clásica excepto porque no aceptamos el tercio excluso.
3. No es cierto que para toda A, se tenga (A o no A).

Heyting y Kolmogorov describen las implicaciones de la lógica intuicionista $A \rightarrow B$ como funciones que transforman demostraciones de A en transformaciones de B . Bajo esta interpretación el cálculo lambda es un sistema de demostraciones de la lógica proposicional intuicionista.

$$\frac{}{\Gamma \vdash * : 1} \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \quad \frac{\Gamma \vdash m : A \times B}{\Gamma \vdash \pi_1 m : A} \quad \frac{\Gamma \vdash m : A \times B}{\Gamma \vdash \pi_2 m : B}$$
$$\frac{\Gamma, x : A \vdash m : B}{\Gamma \vdash \lambda x. m : A \rightarrow B} \quad \frac{\Gamma \vdash f : A \rightarrow B \quad \Gamma \vdash a : A}{\Gamma \vdash f a : B}$$
$$\frac{\Gamma \vdash m : 0}{\Gamma \vdash \text{abort}_A m : A} \quad \frac{\Gamma \vdash a : A}{\Gamma \vdash \text{inl } a : A + B} \quad \frac{\Gamma \vdash b : B}{\Gamma \vdash \text{inr } b : A + B}$$
$$\frac{\Gamma \vdash m : A + B \quad \Gamma, a : A \vdash n : C \quad \Gamma, b : B \vdash p : C}{\Gamma \vdash (\text{case } m \text{ of } [a].n; [b].p) : C}$$

Teorema (Curry, Howard)

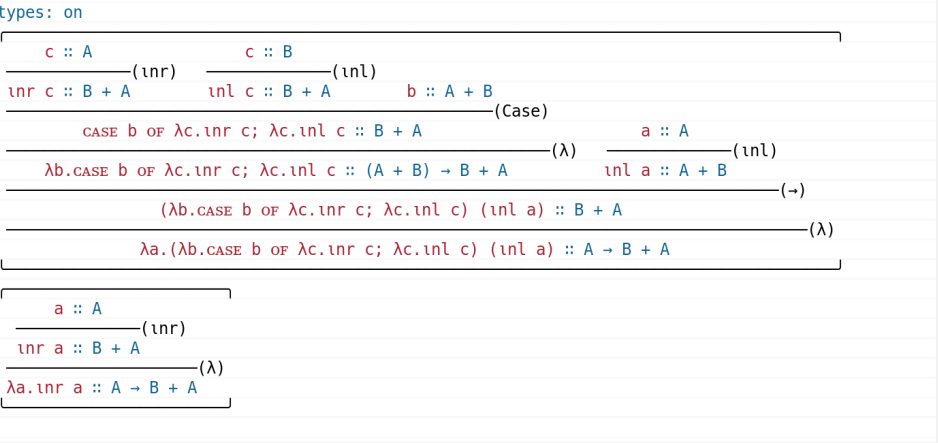
Las proposiciones son tipos, las demostraciones sus elementos. Evaluar elementos equivale a simplificar demostraciones manteniendo su significado.

Philip Wadler. “Propositions As Types”. En: Commun. ACM 58.12 (nov. de 2015), págs. 75-84. ISSN: 0001-0782. DOI: 10.1145/2699407. URL: <http://doi.acm.org/10.1145/2699407>.

1. Resulta que en los años 30 Heyting y Kolmogorov propusieron una lectura de la lógica intuicionista tomando implicaciones como funciones.
2. La lógica intuicionista funciona igual que la clásica excepto porque no aceptamos el tercio excluso.
3. No es cierto que para toda A, se tenga (A o no A).

```
1 :types on
2
3 # Draws the deduction tree
4 @ \a.((\c.Case c Of inr; inl)(INL a))
5
6 # Simplifies the deduction tree
7 @@ \a.((\c.Case c Of inr; inl)(INL a))
```

evaluate



1. Implementamos esto en Haskell.
2. Esta es la versión en Javascript del intérprete. Transpilé una a Javascript y la otra es una adaptación a Jupyter Notebook.
3. La adaptación a Jupyter se ha usado en las clases de lógica y programación para enseñar cálculo lambda.
4. Respecto al ejemplo, las expresiones lambda son demostraciones, contienen en su estructura el árbol de derivación.
5. Y simplificarlas equivale a simplificar la demostración por el método de Gentzen.

Una **adjunción** $F \dashv G$ entre categorías \mathcal{C} y \mathcal{D} es un par de funtores $F\colon \mathcal{C} \rightarrow \mathcal{D}$ y $G\colon \mathcal{D} \rightarrow \mathcal{C}$ con una biyección natural $\varphi\colon \operatorname{hom}(FX, Y) \cong \operatorname{hom}(X, GY)$.

$$\frac{FX \xrightarrow{f} Y}{X \xrightarrow{\varphi(f)} GY}$$

1. Las adjunciones son biyecciones naturales entre morfismos determinados por dos funtores. Las notaremos como inferencias lógicas bidireccionales.
2. El ejemplo común es el de los grupos libres y los conjuntos subyacentes, el homomorfismo de grupos queda determinado por las imágenes de los generadores
3. Los productos y los coproductos se pueden ver como adjuntos al funtor diagonal si los escribimos correctamente.

Una **adjunción** $F \dashv G$ entre categorías \mathcal{C} y \mathcal{D} es un par de funtores $F: \mathcal{C} \rightarrow \mathcal{D}$ y $G: \mathcal{D} \rightarrow \mathcal{C}$ con una biyección natural $\varphi: \text{hom}(FX, Y) \cong \text{hom}(X, GY)$.

$$\frac{FX \xrightarrow{f} Y}{X \xrightarrow{\varphi(f)} GY}$$

Ejemplo (Grupos)

El funtor que crea grupos libres $F: \mathbf{Grp} \rightarrow \mathbf{Set}$, es el adjunto al funtor que a cada grupo le asigna su conjunto subyacente $\lfloor - \rfloor: \mathbf{Set} \rightarrow \mathbf{Grp}$. El homomorfismo queda determinado por la elección de imágenes de los generadores.

$$\frac{FA \xrightarrow{\phi} M}{A \xrightarrow{f} \lfloor M \rfloor}$$

Ejemplo (Productos y coproductos)

Los productos y los coproductos pueden definirse sin hacer referencia a conjuntos como adjuntos $+ \dashv \Delta \dashv \times$ para Δ el funtor diagonal.

$$\frac{X, X \longrightarrow Y, Z}{X \longrightarrow Y \times Z}$$

$$\frac{X + Y \longrightarrow Z}{X, Y \longrightarrow Z, Z}$$

1. Las adjunciones son biyecciones naturales entre morfismos determinados por dos funtores. Las notaremos como inferencias lógicas bidireccionales.
2. El ejemplo común es el de los grupos libres y los conjuntos subyacentes, el homomorfismo de grupos queda determinado por las imágenes de los generadores
3. Los productos y los coproductos se pueden ver como adjuntos al funtor diagonal si los escribimos correctamente.

Una **adjunción** $F \dashv G$ entre categorías \mathcal{C} y \mathcal{D} es un par de funtores $F\colon \mathcal{C} \rightarrow \mathcal{D}$ y $G\colon \mathcal{D} \rightarrow \mathcal{C}$ con una biyección natural $\varphi\colon \operatorname{hom}(FX, Y) \cong \operatorname{hom}(X, GY)$.

$$\frac{FX \xrightarrow{f} Y}{X \xrightarrow{\varphi(f)} GY}$$

Ejemplo (Grupos)

El funtor que crea grupos libres $F\colon \mathbf{Grp} \rightarrow \mathbf{Set}$, es el adjunto al funtor que a cada grupo le asigna su conjunto subyacente $\lfloor - \rfloor\colon \mathbf{Set} \rightarrow \mathbf{Grp}$. El homomorfismo queda determinado por la elección de imágenes de los generadores.

$$\frac{FA \xrightarrow{\phi} M}{A \xrightarrow{f} \lfloor M \rfloor}$$

Ejemplo (Productos y coproductos)

Los productos y los coproductos pueden definirse sin hacer referencia a conjuntos como adjuntos $+ \dashv \Delta \dashv \times$ para Δ el funtor diagonal.

$$\frac{X \rightarrow Y \qquad X \rightarrow Z}{X \longrightarrow Y \times Z}$$

$$\frac{X + Y \longrightarrow Z}{X \rightarrow Z \qquad Y \rightarrow Z}$$

- 1. Las adjunciones son biyecciones naturales entre morfismos determinados por dos funtores. Las notaremos como inferencias lógicas bidireccionales.
- 2. El ejemplo común es el de los grupos libres y los conjuntos subyacentes, el homomorfismo de grupos queda determinado por las imágenes de los generadores
- 3. Los productos y los coproductos se pueden ver como adjuntos al funtor diagonal si los escribimos correctamente.

Una **categoría cartesiana** \mathcal{C} es aquella en la que el funtor terminal $\ast: \mathcal{C} \rightarrow 1$, el funtor diagonal $\Delta: \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ y sus funtores producto $(- \times A): \mathcal{C} \rightarrow \mathcal{C}$ tienen adjuntos derechos. Los llamamos **unidad**, **producto** y **exponencial**.

$$\frac{\ast \longrightarrow \ast}{C \xrightarrow{!} 1} \qquad \frac{C, C \xrightarrow{f,g} A, B}{C \xrightarrow{\langle f,g \rangle} A \times B} \qquad \frac{C \times A \xrightarrow{f} B}{C \xrightarrow{\tilde{f}} B^A}$$

1. En las categorías cartesianas pedimos que existan tres adjunciones derechas, al terminal, a la diagonal y al producto.
2. En conjuntos, el conjunto de un elemento, producto cartesiano y los conjuntos de funciones.
3. Lo interesante es que se corresponden a las reglas del cálculo lambda simplemente tipado.
4. El cálculo lambda es un lenguaje interno de las categorías cartesianas. Esto se hace formal en los resultados de Lambek, donde la equivalencia tiene un significado formal específico.

Joachim Lambek y Philip J Scott. Introduction to higher-order categorical logic. Vol. 7. Cambridge University Press, 1988.

Una **categoría cartesiana** \mathcal{C} es aquella en la que el funtor terminal $*$: $\mathcal{C} \rightarrow 1$, el funtor diagonal Δ : $\mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ y sus funtores producto $(- \times A)$: $\mathcal{C} \rightarrow \mathcal{C}$ tienen adjuntos derechos. Los llamamos **unidad**, **producto** y **exponencial**.

$$\frac{* \longrightarrow *}{C \xrightarrow{!} 1} \qquad \frac{C, C \xrightarrow{f,g} A, B}{C \xrightarrow{\langle f,g \rangle} A \times B} \qquad \frac{C \times A \xrightarrow{f} B}{C \xrightarrow{\tilde{f}} B^A}$$

Si interpretamos los tipos y contextos como objetos y los elementos como morfismos, son las reglas para el cálculo lambda tipado.

$$\frac{}{\Gamma \vdash * : 1} \qquad \frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B}{\Gamma \vdash \langle a, b \rangle : A \times B} \qquad \frac{\Gamma, a : A \vdash b : B}{\Gamma \vdash (\lambda a. b) : A \rightarrow B}$$

Teorema (Lambek)

Hay una equivalencia entre categorías cartesianas cerradas y teorías sobre el cálculo lambda $CCC \simeq \lambda Th$.

1. En las categorías cartesianas pedimos que existan tres adjunciones derechas, al terminal, a la diagonal y al producto.
2. En conjuntos, el conjunto de un elemento, producto cartesiano y los conjuntos de funciones.
3. Lo interesante es que se corresponden a las reglas del cálculo lambda simplemente tipado.
4. El cálculo lambda es un lenguaje interno de las categorías cartesianas. Esto se hace formal en los resultados de Lambek, donde la equivalencia tiene un significado formal específico.

Joachim Lambek y Philip J Scott. Introduction to higher-order categorical logic. Vol. 7. Cambridge University Press, 1988.

Teorema (Lawvere)

En una categoría cartesiana cerrada, si existe un $d : A \rightarrow B^A$ sobreyectivo en puntos, cada $f : B \rightarrow B$ tiene un punto fijo $b : B$, tal que $f(b) = b$.

Demostración. Por sobreyectividad, existe $d\ x = \lambda a.f(d\ a\ a)$. Entonces $d\ x\ x \equiv (\lambda a.f\ (d\ a\ a))\ x \equiv f\ (d\ x\ x)$ es un punto fijo. \square

1. Eso quiere decir que podemos usar el cálculo lambda para demostrar en categorías cartesianas.
2. Pensé en usar como ejemplo un artículo de Lawvere sobre argumentos diagonales en categorías cartesianas.
3. La demostración es muy sencilla, pero tiene consecuencias muy fuertes.
4. Sobreyectiva da puntos fijos, así que si no hay punto fijo no puede ser sobreyectiva.
5. Cantor es el caso particular de conjuntos.
6. Russell se obtiene considerando pertenencia.
7. El teorema de Tarski-Gödel es un caso particular si consideramos una categoría sintáctica donde la consistencia es que haya un morfismo de los valores de verdad sin puntos fijos.

F. William Lawvere. “Diagonal arguments and Cartesian Closed Categories”. En: Reprints in Theory and Applications of Categories (2016), págs. 1-13.

Teorema (Lawvere)

En una categoría cartesiana cerrada, si existe un $d : A \rightarrow B^A$ sobreyectivo en puntos, cada $f : B \rightarrow B$ tiene un punto fijo $b : B$, tal que $f(b) = b$.

Demostración. Por sobreyectividad, existe $d\ x = \lambda a.f(d\ a\ a)$. Entonces $d\ x\ x \equiv (\lambda a.f\ (d\ a\ a))\ x \equiv f\ (d\ x\ x)$ es un punto fijo. \square

Ejemplo (Cantor)

Para todo A , el conjunto 2^A tiene mayor cardinalidad.

Ejemplo (Russell)

Tomar toda colección como conjunto lleva a inconsistencia.

Ejemplo (Combinador de punto fijo)

El cálculo lambda sin tipos puede verse como una teoría sobre el cálculo tipado. Todo combinador tiene un punto fijo.

F. William Lawvere. “Diagonal arguments and Cartesian Closed Categories”. En: Reprints in Theory and Applications of Categories (2016), págs. 1-13.

Ejemplo (Tarski, Gödel)

Una teoría consistente no puede expresar su propia verdad. Consideramos objetos $2, A^0, A^1, A^2, \dots$ y los morfismos entre ellos son expresiones de la teoría. La teoría es **consistente** si existe $\text{not} : 2 \rightarrow 2$ sin puntos fijos. La **verdad** es expresable si existe $\text{truth} : A \times A \rightarrow 2$ tal que para cada $\varphi : A \rightarrow 2$ existe un $g : A$ **número de Gödel** tal que $\text{truth}(g, a) = \varphi(a)$.

1. Eso quiere decir que podemos usar el cálculo lambda para demostrar en categorías cartesianas.
2. Pensé en usar como ejemplo un artículo de Lawvere sobre argumentos diagonales en categorías cartesianas.
3. La demostración es muy sencilla, pero tiene consecuencias muy fuertes.
4. Sobreyectiva da puntos fijos, así que si no hay punto fijo no puede ser sobreyectiva.
5. Cantor es el caso particular de conjuntos.
6. Russell se obtiene considerando pertenencia.
7. El teorema de Tarski-Gödel es un caso particular si consideramos una categoría sintáctica donde la consistencia es que haya un morfismo de los valores de verdad sin puntos fijos.

Un **álgebra inicial** sobre un funtor viene dada por una construcción con la siguiente propiedad universal. Un ejemplo son los números naturales.

$$\begin{array}{ccc} FX & \overset{Fh}{\dashrightarrow} & FY \\ \downarrow \mu & & \downarrow \nu \\ X & \overset{h}{\dashrightarrow} & Y \end{array} \qquad \begin{array}{ccc} 1 + \mathbb{N} & \longrightarrow & 1 + X \\ \downarrow \langle 0, \text{succ} \rangle & & \downarrow \langle x, f \rangle \\ \mathbb{N} & \overset{\varphi}{\longrightarrow} & X \end{array}$$

Lo interesante es que así ganamos una forma de expresar los naturales y los principios de inducción. El cálculo lambda que se obtiene cuando añadimos los naturales se llama **System T** y fue desarrollado por Gödel.

$$\begin{array}{ccc} 1 + \mathbb{N} & \longrightarrow & 1 + \text{hom}(\mathbb{N}, \mathbb{N}) \\ \downarrow \langle 0, \text{succ} \rangle & & \downarrow \langle \text{id}, \text{succ} \circ - \rangle \\ \mathbb{N} & \overset{+}{\longrightarrow} & \text{hom}(\mathbb{N}, \mathbb{N}) \end{array}$$

Nos da $0 + m = (m) = m$ y además $\text{succ}(n) + m = (\text{succ} \circ (n + _))(m) = \text{succ}(n + m)$.

1. Una vez que tenemos toda la estructura, queremos incluir los números naturales.
2. Al construirlos como álgebra inicial, ganamos el principio de inducción como propiedad universal.
3. Por ejemplo, definimos la suma al estilo de Peano por inducción.

Los productos fibrados determinan **sustituciones**. Un caso particular es el **debilitamiento lógico**. Determina un funtor entre **sobrecategorías**.

$$\begin{array}{ccc} \{P(\pi(a,b))\} & \longrightarrow & \{P(a)\} \\ \downarrow & & \downarrow \\ A \times B & \xrightarrow{\pi} & A \end{array}$$

Este debilitamiento tiene dos adjuntos en sobrecategorías, $\exists \dashv \pi \dashv \forall$, que son los cuantificadores lógicos.

$$\begin{array}{ccc} & A \times B & \\ \nearrow & & \nwarrow \\ \{P(\pi(a,b))\} & \xrightarrow{\quad} & \{Q(a,b)\} \\ \hline \{P(a)\} & \xrightarrow{\quad} & \{\forall b \in B, Q(a,b)\} \\ \searrow & & \swarrow \\ & A & \end{array}$$

Teorema
Si existen los productos fibrados, la adjunción izquierda, **existe**, viene dada por composición con π . Si además existe la adjunción derecha, **para todo**, llamamos a la categoría **localmente cartesiana cerrada**.

1. Hasta aquí hemos desarrollado estructura lógica proposicional, el siguiente paso es incluir cuantificadores.
2. Tomamos objetos en sobrecategorías, un objeto es un morfismo hacia un objeto base.
3. La sustitución y el debilitamiento lógico son funtores entre sobrecategorías.
4. Lo interesante es que el debilitamiento da los cuantificadores como adjuntos.
5. Si demuestro una implicación y el antecedente no depende de b, lo estoy demostrando para todo b.
6. Si demuestro una implicación y el consecuente no depende de b, lo estoy demostrando con que exista b.
7. Uno lo tenemos siempre, el otro podemos pedir su existencia a la estructura de la categoría.

Los productos fibrados determinan **sustituciones**. Un caso particular es el **debilitamiento lógico**. Determina un funtor entre **sobrecategorías**.

$$\begin{array}{ccc} \{P(\pi(a,b))\} & \longrightarrow & \{P(a)\} \\ \downarrow & & \downarrow \\ A \times B & \xrightarrow{\pi} & A \end{array}$$

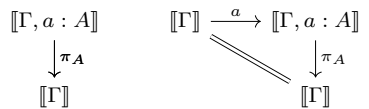
Este debilitamiento tiene dos adjuntos en sobrecategorías, $\exists \dashv \pi \dashv \forall$, que son los cuantificadores lógicos.

$$\begin{array}{ccc} & A \times B & \\ \nearrow & & \nwarrow \\ \{P(a,b)\} & \xrightarrow{\quad} & \{Q(\pi(a,b))\} \\ \hline \{ \exists b \in B: P(a) \} & \xrightarrow{\quad} & \{Q(a)\} \\ \searrow & & \swarrow \\ & A & \end{array}$$

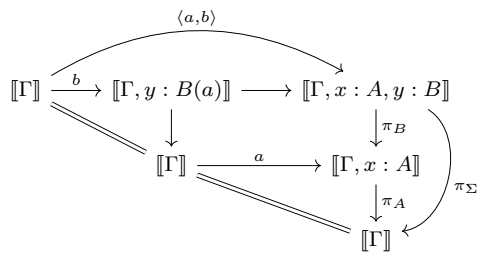
Teorema
Si existen los productos fibrados, la adjunción izquierda, **existe**, viene dada por composición con π . Si además existe la adjunción derecha, **para todo**, llamamos a la categoría **localmente cartesiana cerrada**.

1. Hasta aquí hemos desarrollado estructura lógica proposicional, el siguiente paso es incluir cuantificadores.
2. Tomamos objetos en sobrecategorías, un objeto es un morfismo hacia un objeto base.
3. La sustitución y el debilitamiento lógico son funtores entre sobrecategorías.
4. Lo interesante es que el debilitamiento da los cuantificadores como adjuntos.
5. Si demuestro una implicación y el antecedente no depende de b, lo estoy demostrando para todo b.
6. Si demuestro una implicación y el consecuente no depende de b, lo estoy demostrando con que exista b.
7. Uno lo tenemos siempre, el otro podemos pedir su existencia a la estructura de la categoría.

Vamos a construir \exists en el cálculo lambda. Un tipo es un objeto de la sobrecategoría sobre un contexto, $\mathcal{C}/[[\Gamma]]$. Sus elementos son morfismos desde el terminal de la sobrecategoría, $\text{id} : [[\Gamma]] \rightarrow [[\Gamma]]$.

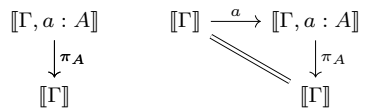


Esta construcción es un **par dependiente** o **tipo Sigma**.

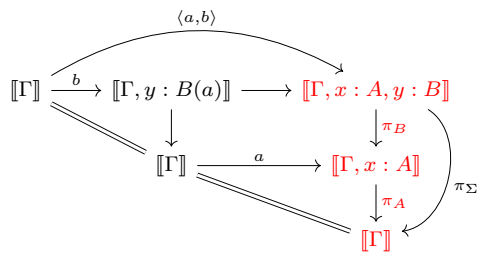


1. Construimos el existencial en cálculo lambda.
2. Los tipos son objetos de la sobrecatgeoría, estos triángulos son elementos, morfismos desde el terminal.
3. Esta composición hemos dicho que es el existencial, damos un elemento abajo, sustituimos por producto fibrado y damos un elemento arriba.
4. Construir una demostración de que existe $a:A$ tal que $B(a)$ es dar un a_0 y luego probar $B(a_0)$.

Vamos a construir \exists en el cálculo lambda. Un tipo es un objeto de la sobrecategoría sobre un contexto, $\mathcal{C}/[[\Gamma]]$. Sus elementos son morfismos desde el terminal de la sobrecategoría, $\text{id} : [[\Gamma]] \rightarrow [[\Gamma]]$.



Esta construcción es un **par dependiente** o **tipo Sigma**.

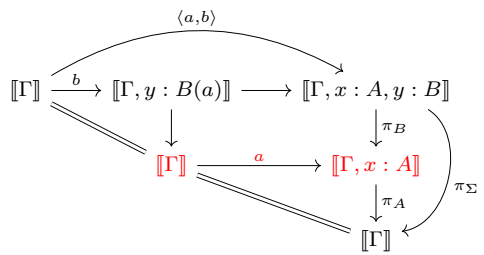


- 1. Construimos el existencial en cálculo lambda.
- 2. Los tipos son objetos de la sobrecatgeoría, estos triángulos son elementos, morfismos desde el terminal.
- 3. Esta composición hemos dicho que es el existencial, damos un elemento abajo, sustituimos por producto fibrado y damos un elemento arriba.
- 4. Construir una demostración de que existe $a:A$ tal que $B(a)$ es dar un a_0 y luego probar $B(a_0)$.

Vamos a construir \exists en el cálculo lambda. Un tipo es un objeto de la sobrecategoría sobre un contexto, $\mathcal{C}/\llbracket \Gamma \rrbracket$. Sus elementos son morfismos desde el terminal de la sobrecategoría, $\text{id} \colon \llbracket \Gamma \rrbracket \rightarrow \llbracket \Gamma \rrbracket$.

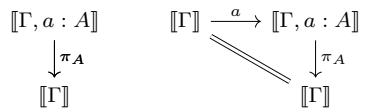
$$\begin{array}{ccc} \llbracket \Gamma, a : A \rrbracket & & \llbracket \Gamma \rrbracket \xrightarrow{a} \llbracket \Gamma, a : A \rrbracket \\ \downarrow \pi_A & & \searrow \parallel \downarrow \pi_A \\ \llbracket \Gamma \rrbracket & & \llbracket \Gamma \rrbracket \end{array}$$

Esta construcción es un **par dependiente** o **tipo Sigma**.

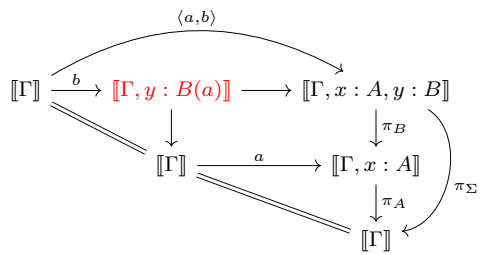


1. Construimos el existencial en cálculo lambda.
2. Los tipos son objetos de la sobrecatgeoría, estos triángulos son elementos, morfismos desde el terminal.
3. Esta composición hemos dicho que es el existencial, damos un elemento abajo, sustituimos por producto fibrado y damos un elemento arriba.
4. Construir una demostración de que existe $a:A$ tal que $B(a)$ es dar un a_0 y luego probar $B(a_0)$.

Vamos a construir \exists en el cálculo lambda. Un tipo es un objeto de la sobrecategoría sobre un contexto, $\mathcal{C}/[[\Gamma]]$. Sus elementos son morfismos desde el terminal de la sobrecategoría, $\text{id} : [[\Gamma]] \rightarrow [[\Gamma]]$.

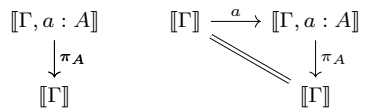


Esta construcción es un **par dependiente** o **tipo Sigma**.

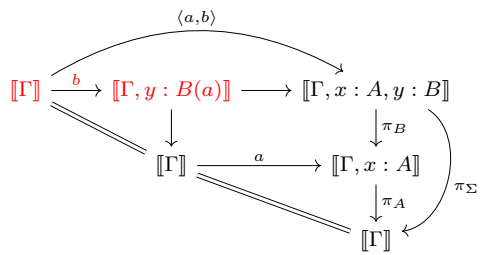


- 1. Construimos el existencial en cálculo lambda.
- 2. Los tipos son objetos de la sobrecatgeoría, estos triángulos son elementos, morfismos desde el terminal.
- 3. Esta composición hemos dicho que es el existencial, damos un elemento abajo, sustituimos por producto fibrado y damos un elemento arriba.
- 4. Construir una demostración de que existe $a:A$ tal que $B(a)$ es dar un a_0 y luego probar $B(a_0)$.

Vamos a construir \exists en el cálculo lambda. Un tipo es un objeto de la sobrecategoría sobre un contexto, $\mathcal{C}/[[\Gamma]]$. Sus elementos son morfismos desde el terminal de la sobrecategoría, $\text{id} : [[\Gamma]] \rightarrow [[\Gamma]]$.



Esta construcción es un **par dependiente** o **tipo Sigma**.



- 1. Construimos el existencial en cálculo lambda.
- 2. Los tipos son objetos de la sobrecatgeoría, estos triángulos son elementos, morfismos desde el terminal.
- 3. Esta composición hemos dicho que es el existencial, damos un elemento abajo, sustituimos por producto fibrado y damos un elemento arriba.
- 4. Construir una demostración de que existe $a:A$ tal que $B(a)$ es dar un a_0 y luego probar $B(a_0)$.

La construcción del **cuantificador universal** son las **funciones dependientes** o **tipos Π** . Es directa desde la adjunción suponiendo que estamos en una categoría localmente cartesiana cerrada.

1. En el caso del cuantificador universal es más fácil porque simplemente asumimos que existe como un adjunto.
2. Se pueden ver en cálculo lambda como funciones, donde la aplicación sale adjunta a la identidad.

$$\frac{\begin{array}{ccc} & \curvearrowright & \\ \llbracket \Gamma, A \rrbracket & \xrightarrow{b} & \llbracket \Gamma, A, B \rrbracket \\ & \curvearrowleft & \\ & \llbracket \Gamma, A \rrbracket & \end{array}}{\llbracket \Gamma \rrbracket \xrightarrow{(\lambda a. b)} \llbracket \Gamma, \prod_{a:A} B \rrbracket} \qquad \frac{\begin{array}{ccc} & \curvearrowright & \\ \llbracket \Gamma, A, \prod_{a:A} B \rrbracket & \xrightarrow{app} & \llbracket \Gamma, A, B \rrbracket \\ & \curvearrowleft & \\ & \llbracket \Gamma, A, \prod_{a:A} B \rrbracket & \end{array}}{\llbracket \Gamma, \prod_{a:A} B \rrbracket \xrightarrow{id} \llbracket \Gamma, \prod_{a:A} B \rrbracket}$$

Nótese cómo la imagen de la identidad bajo la adjunción es la aplicación de una función sobre un elemento.

Robert AG Seely. “Locally cartesian closed categories and type theory”. En: Mathematical proceedings of the Cambridge philosophical society. Vol. 95. 1. Cambridge University Press. 1984, págs. 33-48.

1. Cálculo lambda no tipado

2. Cálculo lambda tipado

3. Categorías cartesianas

4. Categorías localmente cartesianas

5. Teoría de tipos

6. Matemática constructivista

1. De toda esta estructura obtenemos una serie de reglas que modelan los tipos dependientes en lenguajes de programación.
2. Por ejemplo, dar un elemento del existencial es dar un natural y luego un vector de esa longitud.
3. Dar un elemento del universal es dar una función que construya un vector de la longitud pedida.

Obtenemos las siguientes reglas para un sistema de tipos dependiente.

$$\frac{\Gamma \vdash a : A \quad \Gamma \vdash b : B[a/x]}{\Gamma \vdash \langle a, b \rangle : \sum_{x:A} B} \quad \frac{\Gamma \vdash m : \sum_{x:A} C}{\Gamma \vdash \mathbf{fst}(m) : A} \quad \frac{\Gamma \vdash m : \sum_{x:A} C}{\Gamma \vdash \mathbf{snd}(m) : C[(m)/a]}$$
$$\frac{\Gamma, a : A \vdash b : B}{\Gamma \vdash (\lambda a. b) : \prod_{a:A} B} \quad \frac{\Gamma \vdash a : A \quad \Gamma \vdash f : \prod_{a:A} B}{\Gamma \vdash f \ a : B(a)}$$

Ejemplo: diferencia entre vectores y vectores uniformes.

$$v : \sum_{n:\mathbb{N}} \mathbf{Vect}(n) \quad \text{y} \quad w : \prod_{n:\mathbb{N}} \mathbf{Vect}(n).$$

Per Martin-Löf. “An intuitionistic theory of types: Predicative part”. En: Studies in Logic and the Foundations of Mathematics. Vol. 80. Elsevier, 1975, págs. 73-118.

```
-- Definition of the sigma type as a record with two constructors.  
-- Its elimination principle can be directly derived from the  
-- definition.
```

```
record  $\Sigma$  (S : Set) (T : S  $\rightarrow$  Set) : Set where
```

```
  constructor _,-
```

```
  field
```

```
    fst : S
```

```
    snd : T fst
```

```
open  $\Sigma$  public
```

```
first : {A : Set} {B : A  $\rightarrow$  Set} {C : Set}  $\rightarrow \Sigma$  A B  $\rightarrow$  A
```

```
first (a , b) = a
```

```
second : {A : Set} {B : A  $\rightarrow$  Set} {C : Set}  $\rightarrow$  (r :  $\Sigma$  A B)  $\rightarrow$  B (fst r)
```

```
second (a , b) = b
```

```
- 1.8k Base.agda Agda :Checked Git:master Mod | unix | 57: 0 44%
```

```
% 0 *All Done* AgdaInfo | unix | 1: 0 All
```

1. Lo interesante es que ahora toda esta estructura lógica nos la podemos llevar a un lenguaje de programación.
2. Podemos implementar así el existencial.
3. Podríamos intentar hacer matemáticas dentro de un lenguaje, pero necesitamos más estructura.

La **igualdad** será el tipo dado por la diagonal $\Delta: A \rightarrow A \times A$. La propiedad universal del producto fibrado nos da una equivalencia entre los siguientes diagramas con la diagonal.

$$\begin{array}{ccc} \Delta^* C & \longrightarrow & C \\ \tilde{k} \downarrow & & \downarrow \pi \\ A & \xrightarrow{\Delta} & A \times A \end{array}$$

$$\begin{array}{ccc} A & \overset{k}{\dashrightarrow} & C \\ \Delta \searrow & & \swarrow \pi \\ & A \times A & \end{array}$$

Cuando lo interpretamos en tipos, a la izquierda tenemos un $x : A \vdash c : C(x, x)$ y a la derecha tenemos $x : A, y : A, p : x = y \vdash c : C(x, y)$. Esto nos da una regla para usar la igualdad, que en teoría de tipos se llama **eliminador J**.

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : A \qquad \Gamma, x : A \vdash c : C(x, x) \qquad \Gamma \vdash p : a = b}{\Gamma \vdash J_C(c, p) : C(a, b)}$$

1. Hay una idea de Lawvere que implementa la igualdad usando que los dos siguientes diagramas equivalen.
2. Esto nos da que tenemos la igualdad en el caso de reflexividad y que si probamos algo para la diagonal de una variable, lo probamos para cualesquiera a y b iguales.
3. Este principio se llama eliminador J.

F. William Lawvere. “Equality in hyperdoctrines and comprehension schema as an adjoint functor”. En: Applications of Categorical Algebra 17 (1970), págs. 1-14.


```

-- Equality is defined as an inductive type. Its induction principle
-- is the J-eliminator.
data _==_ {ℓ} {A : Type ℓ} : A → A → Type ℓ where
  refl : (a : A) → a == a

-- Composition of paths
infixl 50 _·_
_·_ : ∀{ℓ} {A : Type ℓ} {a b c : A} → a == b → b == c → a == c
refl a · q = q

```

1. Podemos implementar el eliminador J como una familia inductivamente generada de tipos.

- 1.9k agda-hott/Base.agda Agda :Checked  unix | 70: 0 Bottom

% 0 *All Done* AgdaInfo  unix | 1: 0 All

Un **subobjeto clasificador** es un Ω con un monomorfismo $\text{true} : 1 \rightarrow \Omega$ tal que, para todo monomorfismo $m : \llbracket \Gamma, P \rrbracket \rightarrow \llbracket \Gamma \rrbracket$, existe un único χ tal que el siguiente diagrama es un producto fibrado.

$$\begin{array}{ccc} \llbracket \Gamma, x : P \rrbracket & \longrightarrow & 1 \\ \downarrow & & \downarrow \text{true} \\ \llbracket \Gamma \rrbracket & \overset{\chi_P}{\dashrightarrow} & \Omega \end{array}$$

Los tipos dados por un monomorfismo se llaman **proposiciones** y se pueden ver como elementos del tipo Ω .

$$\frac{\Gamma \vdash P : \Omega \quad \Gamma \vdash a : P \quad \Gamma \vdash b : P}{\Gamma \vdash \text{isProp}_P(a, b) : a = b}$$

Además, mediante adjunciones hay formas de **truncar** cada tipo A en una proposición $|A|$.

1. El subobjeto clasificador da una construcción categórica del conjunto de valores de verdad.
2. En conjuntos sería el 2, y es el que nos permitiría hacer conjuntos potencia.
3. Aquí, es un tipo de los tipos dados por un monomorfismo, es decir, de los de un solo elemento.
4. Con esto tenemos suficiente para hacer matemática constructiva.

Con toda la estructura considerada podemos interpretar fundaciones categóricas de las matemáticas dentro de un lenguaje de programación. Un ejemplo es la **Elementary Theory of the Category of Sets** de W. Lawvere. Axiomas:

- una categoría localmente cartesiana cerrada con todos los límites finitos, el álgebra inicial de $1 + X$ y un subobjeto clasificador (un **topos con un objeto de números naturales**);
- **punteada**, para cada $f, g: A \rightarrow B$, hay igualdad $f = g$ si y sólo si $f(a) = g(a)$ para cualquier $a: 1 \rightarrow A$;
- cumpliendo el **axioma de elección**, los morfismos sobreyectivos en puntos tienen una sección.

$$\left(\prod_{(a:A)} \left\| \sum_{(b:B)} f(b) = a \right\| \right) \rightarrow \left\| \sum_{(g:A \rightarrow B)} \prod_{(a:A)} f(g(a)) = a \right\|$$

1. La matemática constructiva no asume el tercio excluso.
2. Algunos resultados obvios en otro caso son aquí independientes.
3. El subconjunto de un conjunto finito no tiene por qué ser finito, un espacio vectorial no tiene por qué tener base.
4. Pero hay casos particulares donde podemos recuperar la matemática clásica.
5. La teoría elemental de conjuntos de Lawvere toma toda la estructura hasta el momento.
6. Añade el axioma de que las funciones quedan definidas por sus imágenes.
7. Y el axioma de elección, que desde fuera es simplemente ver que las sobreyecciones pueden dar la vuelta.

F. William Lawvere. “An elementary theory of the category of sets”. En: Proceedings of the national academy of sciences 52.6 (1964), págs. 1506-1511.

```
-- We internally postulate well-pointedness.
postulate
  wellPointed : {A : Set} {B : A → Set} → {f g : (a : A) → B a}
    → ((x : A) → f x ≡ g x) → f ≡ g

-- We internally postulate the Axiom of Choice.
postulate
  AxiomOfChoice : {A : Set} {B : Set} {R : A → B → Set}
    → ((a : A) → (∃ b ∈ B , (R a b)))
    -----
    → (∃ g ∈ (A → B), ((a : A) → R a (g a)))
```



- 2.5k **Etcs.agda** Agda :Checked Git:master Mod  unix | 35: 0 18%

% 0 ***All Done*** AgdaInfo  unix | 1: 0 All

Wrote /home/mario/projects/ctlc/agda-mltt/Etcs.agda

1. Podemos implementar esto en Agda.

1. Con el axioma de elección volvemos a la matemática clásica.
2. Aunque el axioma de elección suele involucrar cardinales grandes, aquí se usa para conjuntos muy pequeños.
3. La demostración, como se puede ver, es sencilla.

Teorema (Diaconescu)

El axioma de elección implica el tercio excluso.

Demostración.

Dada P , definimos $U = \{x \in \{0, 1\} \mid (x = 0) \vee P\}$ y $V = \{x \in \{0, 1\} \mid (x = 1) \vee P\}$, cada no vacío. Por elección, existe $f: \{U, V\} \rightarrow U \cup V$ tal que $f(U) \in U$ y $f(V) \in V$. Decidimos si $f(U)$ y $f(V)$ son 0 o no por inducción. Si $f(U) = 1$ o $f(V) = 0$, entonces P es cierto; y si $f(U) = 0$ y además $f(V) = 1$, tendríamos $\neg P$, porque si P , entonces $U = V$, luego $0 = f(U) = f(V) = 1$. □

Así que al aceptar axioma de elección, hemos vuelto a la matemática clásica. Esto es una posible vía, pero no la única.

Radu Diaconescu. “Change of base for toposes with generators”. En: Journal of Pure and Applied Algebra 6.3 (1975), págs. 191-218.

postulate

```
AxiomOfChoice : {A : Set} {B : Set} {R : A → B → Set}
→ ((a : A) → (∃ b ∈ B , (R a b)))
-----
→ (∃ f ∈ (A → B), ((a : A) → R a (f a)))
```

```
LawOfExcludedMiddle : {P : Set} → P ∨ ¬ P
```

```
LawOfExcludedMiddle {P} = Ex-elim
```

```
(AxiomOfChoice ∧ { (Q , q) → Ex-elim q Ex-isProp ∧ { (u , v) → u , v } })
V-isProp ∧ { (f , α) → byCases f α }
```

where

```
A : Set
```

```
A = Σ (Bool → Set) (λ Q → Ex Bool (λ b → Q b))
```

```
R : A → Bool → Set
```

```
R (P , _) b = P b
```

```
U : Bool → Set
```

```
U b = (b ≡ true) ∨ P
```

```
V : Bool → Set
```

```
V b = (b ≡ false) ∨ P
```

* 5.1k Snippets.lagda Agda :Checked Git-master || unix | 170: 0 58%

% 0 *All Done* AgdaInfo || unix | 1: 0 All

1. Nos llevamos la demostración a Agda.

2. El problema es que siguiendo esta vía, perderíamos el contenido computacional al incluir elección.

1. Hay otros modelos constructivos interesantes que tienen interpretación computacional.
2. Es interesante estudiar estos universos, el precio es perder el tercio excluso.

Cada topos da un modelo de matemática constructivista.

- En el **topos de realizabilidad** de Hayland, toda función es computable y podemos estudiar computabilidad sintética y realizabilidad de Kleene
- En el **topos de Dubuc** podemos formalizar los infinitesimales y hacer geometría diferencial sintética.
- En el **topos topológico** de Johnstone, podemos razonar sobre espacios y funciones continuas.

Además, al aceptar tercio excluso perdemos la interpretación computacional de Brower-Heyting-Kolmogorov.

Jaap Van Oosten. Realizability: an introduction to its categorical side. Vol. 152. Elsevier, 2008.

Eduardo J Dubuc. “Integración de campos vectoriales y geometría diferencial sintética”. En: Revista de la Unión Matemática Argentina 35 (1989), págs. 151-162.

Peter T Johnstone. “On a topological topos”. En: Proceedings of the London mathematical society 3.2 (1979), págs. 237-271.

```
-- The following is a construction of the positive real numbers using
-- Dedekind cuts. A cut is a propositional predicate over the rational
-- numbers that determines which rationals are greater or equal than the
-- real number. A real number must be
--   * bounded, meaning that the cut must be inhabited;
--   * rounded, meaning that the cut must be an upper-unbounded and
--     nonclosed interval.
```

```
record  $\mathbb{R}^+$  : Set where
  constructor real
  field
    -- Dedekind cut
    cut : F → Set
    isprop : (q : F) → isProp (cut q)

    bound :  $\exists q \in F$  , cut q
    round1 : (q : F) → cut q →  $\exists p \in F$  , ((p < q  $\equiv$  true) × cut p)
    round2 : (q : F) → ( $\exists p \in F$  , ((p < q  $\equiv$  true) × cut p)) → cut q

open  $\mathbb{R}^+$  {...} public
```

```
* 7.7k Reals.agda Agda :Checked Git-master | unix | 29: 0 3%
```

```
% 0 *All Done* AgdaInfo | unix | 1: 0 All
```

1. Hemos hablado de que es posible hacer matemática en teoría, vamos a verlo en la práctica.
2. He desarrollado dos librerías de Agda, la primera orientada a implementar los números reales positivos.
3. Uso cortes de Dedekind, se pueden demostrar algunas propiedades básicas (conmutatividad de la suma).
4. Pero además, si demostramos ciertas instancias del tercio excluso, ganamos algoritmos que nos permiten hacer cosas como calcular, de forma verificada, dígitos de ciertos reales.
5. Mi ejemplo fue calcular dígitos de la raíz de 2.

```
-- The fundamental group of the circle is the integers. In this
-- proof, univalence is crucial. The next lemma will prove that the
-- equivalence in fact preserves the group structure.
```

```
fundamental-group-of-the-circle :  $\Omega S^1 \text{ base} \simeq \mathbb{Z}$ 
```

```
fundamental-group-of-the-circle = equiv-family base
```


```
preserves-composition :  $\forall n m \rightarrow \text{loops } (n +^z m) == \text{loops } n \cdot \text{loops } m$ 
```

```
preserves-composition n m = z-act+ ( $\Omega\text{-st } S^1 \text{ base}$ ) n m loop
```

```
preserves-inverses :  $\forall n \rightarrow \text{loops } (- n) == \text{inv } (\text{loops } n)$ 
```

```
preserves-inverses n = z-actm ( $\Omega\text{-st } S^1 \text{ base}$ ) n loop
```

1. Podemos ir más allá, si asumimos el axioma de Univalencia de Voevodsky, los infinito-grupoides son un modelo natural.
2. Por hipótesis de Grothendieck, podemos interpretarlos como espacios donde estudiar la homotopía.
3. Es posible probar que el grupo fundamental del círculo son los enteros.

- 4.9k FundGroupCircle.agda Agda :Checked  unix | 136: 0 Bottom

% 0 *All Done* AgdaInfo  unix | 1: 0 All