Igualando proposiciones/tipos isomorfos en las lógicas/lenguajes del λ-cubo

Cristian Sottile

25 de julio de 2022

1. λ-cálculo

El λ-cálculo es una formalización de la computación basada en la construcción y aplicación de funciones. Es introducido en 1936 por Alonzo Church como un lenguage de la lógica, en medio de los intentos por resolver el problema de la decisión (*Entscheidungsproblem*) del Programa de Hilbert, para lo cual se buscaba definir procedimientos "efectivamente calculables" para determinar la verdad o falsedad de cualquier expresión. Esta versión original no es consistente y por lo tanto no sirve como lógica, así que Church agrega un sistema de tipos que impide que un programa se aplique sobre sí mismo, obteniendo una lógica consistente. A la vez constituye un modelo de computación y la base de la programación funcional. Notamos $\lambda^{\to}$ este cálculo con tipos simples, que se define como sigue.

- Términos (variables, abstracción y aplicación)

$$t, s, ... \quad ::= \quad x^A \mid \lambda x^A.t \mid ts$$

- Tipos (primitivo y funciones)

$$A, B, ... \quad ::= \quad \tau \mid A \to B$$

- Juicios de tipado: t tiene tipo A en el contexto Γ

$$\Gamma \vdash t : A$$

- Sistema de tipos (construcción de programas siguiendo las reglas)

$$\frac{}{\Gamma, x^A \vdash x : A}(var) \quad \frac{\Gamma, x^A \vdash t : B}{\Gamma \vdash \lambda x^A.\ t : A \to B}(abs)$$

$$\frac{\Gamma \vdash t : A \to B \quad \Gamma \vdash s : A}{\Gamma \vdash ts : B}(app)$$

- Ejemplo
$$\dfrac{\dfrac{\dfrac{}{y^A, x^A \vdash x : A}\,(var)}{y^A \vdash \lambda x^A.\ x : A \to A}\,(abs) \qquad \dfrac{}{y^A \vdash y : A}\,(var)}{y^A \vdash (\lambda x^A.\ x) y : A}\,(app)$$

- Evaluación: $(\lambda x.\ t)s \rightsquigarrow [x := s]t$ (substitución de variable por argumento)

2. Lógica $\cong$ Computación

La Correspondencia de Curry-Howard relaciona de manera directa a la lógica y a la computación, y nos permite estudiarlas de manera paralela y complementaria. El fragmento implicativo de la lógica proposicional intuicionista se corresponde con el λ-cálculo tipado simple ($\lambda^{\to}$).

- **Proposiciones $\cong$ Tipos**: la gramática de las proposiciones es similar a la de los tipos ($A, B, ... ::= p \mid A \Rightarrow B$), y al ignorar los términos y observando solo los tipos en el sistema presentado, se obtienen exactamente las reglas de dicha lógica.

$$\frac{}{\Gamma, A \vdash A}(ax) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}(\Rightarrow_i) \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}(\Rightarrow_e / MP)$$

- **Pruebas $\cong$ Programas**: todo programa válido (i.e. término tipado) tiene asociada una derivación de su tipo, que por el ítem anterior se corresponde con una prueba lógica.

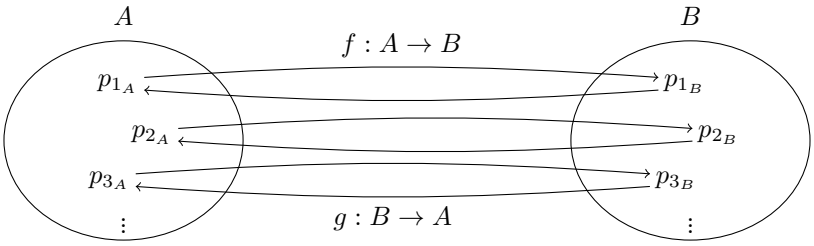
$$\text{e.g.} \qquad \lambda x^A.\ \lambda y^B.\ x \qquad \text{prueba} \qquad A \Rightarrow B \Rightarrow A$$

- **Simplificación $\cong$ Evaluación**: la simplificación de pruebas introduce modificaciones que se corresponden con las que sufren las derivaciones de tipos cuando se evalúa un programa.

Trabajaremos con terminología de computación, pero considerando nociones y consecuencias tanto en computación como en lógica. El cálculo $\lambda^{\to}$ se corresponde con una lógica mínima, y sistemas más complejos corresponden a lógicas más complejas. Utilizaremos terminología de computación, pero considerando nociones y consecuencias tanto en computación como en lógica.

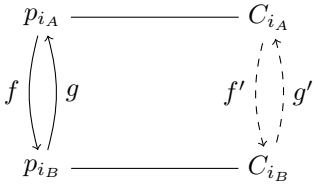
3. Teoría de tipos isomorfos

Dos tipos son isomorfos si permiten que sus programas se transformen entre sí sin pérdida de información. Es decir, si A y B son isomorfos, existen dos funciones f y g que transforman programas de tipo A en programas de tipo B y viceversa, para las que se cumple que compuestas dan la identidad.



Siendo C_{i_A} y C_{i_B} contextos en los que se usarían p_{i_A} y p_{i_B} respectivamente, el isomorfismo nos garantiza que podemos "adaptarlos" mediante f y g para usarlos en el contexto del otro: $C_{i_B}[f\, p_{i_A}]$ y $C_{i_A}[g\, p_{i_B}]$ son combinaciones válidas.

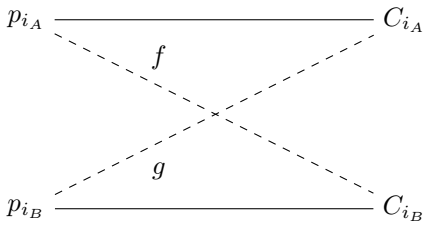
La información que contienen ambos programas es la misma, lo que cambia entre ellos es la disposición de dicha información (podemos pensar en la sintaxis como determinante de la disposición), y esto provoca que los contextos en los que pueden usarse difieran. Si conocemos la disposición de los programas, conocemos en qué contextos pueden usarse, por lo que podremos utilizar funciones f' y g' que transformen los contextos C_A en C_B y viceversa. Esto daría validez a las dos posibles combinaciones restantes: $g'(C_B)[p_A]$ y $f'(C_A)[p_B]$.



En la figura podemos ver líneas horizontales que corresponden al uso tradicional de programas en su contexto, flechas verticales comunes que corresponden a las adaptaciones de programas provistas por el isomorfismo, y flechas verticales punteadas que corresponden a las adaptaciones de contextos.

4. Internalización de isomorfismos

Di Cosmo et. al. caracterizaron los isomorfismos en $\lambda^{\to}$ y algunas de sus principales extensiones. Conociendo todos los isomorfismos de un sistema, resulta natural pensar en automatizar las transformaciones provistas: permitir que en C_{i_A} se use p_{i_B} sin que se aplique "manualmente" f (respectivamente para C_{i_B}, p_{i_A} y g).

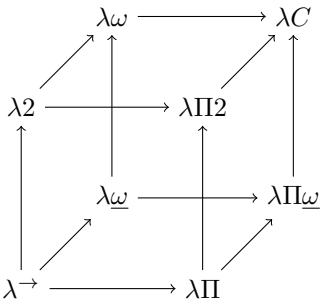


En un sistema así f y g pasan a aplicarse implícitamente (sin intervención de quien programa), y aparecen líneas diagonales que conectan a un programa con los contextos de todos sus correspondientes por isomorfismos.

En este sentido, Díaz-Caro y Dowek introdujeron *System I*, una extensión a $\lambda^{\to}$ en la que los tipos isomorfos se consideran iguales, por lo que las combinaciones $C_{i_A}[p_{i_B}]$ y $C_{i_B}[p_{i_A}]$ son válidas.

5. λ-cubo

Los sistemas de tipos clasifican a los programas formalizando parcialmente propósito y propiedades. En $\lambda^{\to}$ podemos construir funciones de términos en términos y darles un tipo específico. Sistemas más complejos permiten escribir más (o menos) programas y decir más acerca de sus propósitos y propiedades. El λ-cubo relaciona a las principales extensiones de $\lambda^{\to}$.



- **Polimorfismo (↑)**: tipos como parámetro y cuantificación del tipo, e.g. $\mathit{length} : \forall a.[a] \to \mathit{Int}$
- **Constructores de tipos (↗)**: funciones en el nivel de los tipos, e.g. $\mathit{Maybe} : * \to *$
- **Tipos dependientes (→)**: términos como parámetro de los tipos, e.g. vectores cuyo tamaño se expresa en el tipo: $[1, 1, 1] : \mathit{Vec}\ 3$

6. Propuesta de doctorado

Objetivo específico La internalización de isomorfismos es de interés tanto desde la lógica, al permitir que una prueba p_A de una proposición A constituya una prueba de B, para toda B isomorfa a A, como desde la computación, al permitir que un programa p_A de tipo A sea utilizado en donde tradicionalmente se usaría uno de tipo B, para todo B isomorfo a A. Partiendo de *System I*, el objetivo es avanzar hacia la internalización en todos los sistemas del λ-cubo, culminando con λC (Cálculo de Construcciones), que es la base de algunos asistentes de pruebas como Coq.

Trabajo en progreso Actualmente estoy trabajando en la extensión módulo isomorfismos de $\lambda 2$, en particular en la prueba de la propiedad de normalización fuerte (la evaluación de todo programa tipado termina). Paralelamente hemos observado que la extensión hacia constructores de tipos ($\lambda\underline{\omega}$) no presenta mayor interés, por lo que el siguiente paso será hacia tipos dependientes ($\lambda\Pi$).

Objetivo general A futuro buscamos llevar estas ideas a lenguajes/sistemas prácticos como Coq, Agda o Haskell, primero como implementaciones (ya existe una de un *System I* preliminar en Haskell, por Díaz-Caro y Martínez López), y luego incorporándolos como bibliotecas y/o extensiones.