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

Alejandro Díaz-Caro
Director – ICC (UBA/CONICET) & UNQ

Cristian Sottile
Doctorando – ICC (UBA/CONICET)

Pablo E. Martínez López
Codirector – UNQ

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 lenguaje de la lógica, y constituye la base de la programación funcional. Notamos $\lambda^{\to}$ a la versión con tipos simples.

- Términos (variables, abstracción y aplicación)

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

- Tipos (primitivo y funciones) $\qquad A, B, \ldots \quad ::= \quad \tau \quad | \quad A \to B$
- Juicios de tipado: $\qquad \Gamma \vdash t : A \qquad$ (t tiene tipo A en el contexto Γ)
- Sistema de tipos (construcción de programas siguiendo las reglas)

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

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

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.

- **Proposiciones $\cong$ Tipos**:
  - gramáticas similares: $A, B, \ldots ::= p \mid A \Rightarrow B$;
  - ignorar términos en reglas de tipos nos da reglas lógicas:

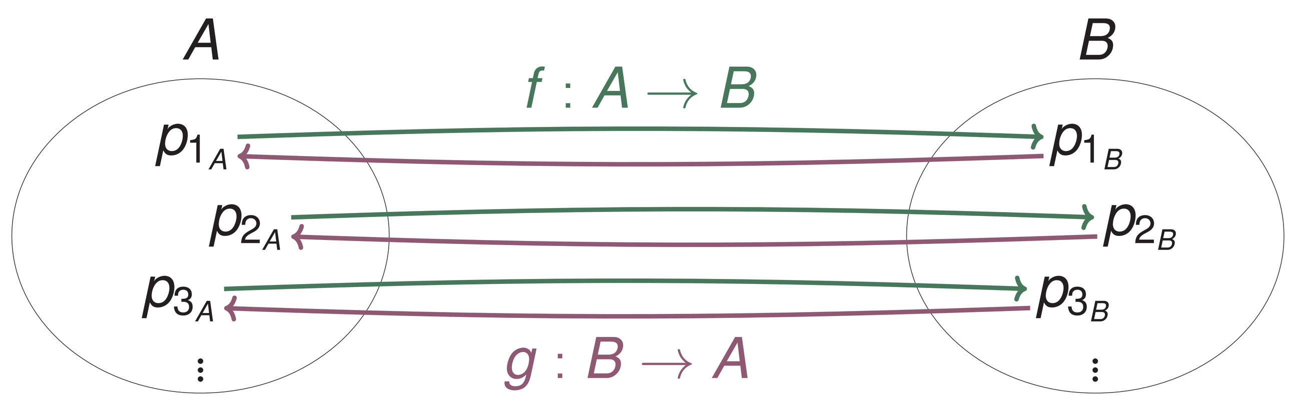
$$\frac{}{\Gamma, A \vdash A}(ax) \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash A \Rightarrow B}(\Rightarrow_i) \qquad \frac{\Gamma \vdash A \Rightarrow B \quad \Gamma \vdash A}{\Gamma \vdash B}(\Rightarrow_e / MP)$$

- **Pruebas $\cong$ Programas**: todo programa válido tiene asociada una derivación de su tipo y por lo tanto una prueba lógica.
  e.g. $\qquad \lambda x^A.\lambda y^B.x \qquad$ prueba $\qquad A \Rightarrow B \Rightarrow A$
- **Simplificación $\cong$ Evaluación**: Los cambios introducidos por ambos procesos en el árbol de derivación son los mismos.

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.

Teoría de tipos isomorfos

Dos tipos son isomorfos si sus programas se pueden transformar entre sí sin pérdida de información, i.e., existen f y g tales que:



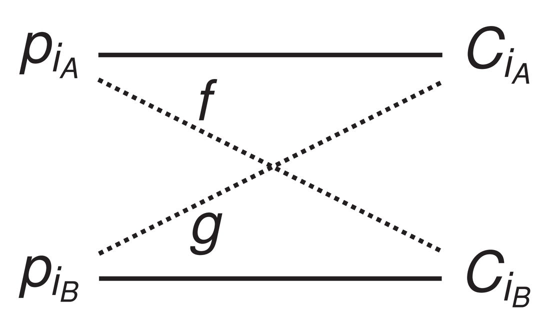
Siendo C_{i_A} y C_{i_B} contextos en los que se usarían p_{i_A} y p_{i_B} respectivamente, el isomorfismo nos garantiza que podemos "adaptarlos" mediante f y g para usarlos en el contexto del otro: $C_{i_B}[f\, p_{i_A}]$ y $C_{i_A}[g\, p_{i_B}]$ son combinaciones válidas.

Las líneas horizontales corresponden al uso tradicional de programas en su contexto, y las flechas verticales a las adaptaciones provistas por el isomorfismo.

$$\begin{tikzcd} p_{i_A} \arrow[r, dash] \arrow[d, bend right, "f"'] & C_{i_A} \\ p_{i_B} \arrow[r, dash] \arrow[u, bend right, "g"'] & C_{i_B} \end{tikzcd}$$

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).



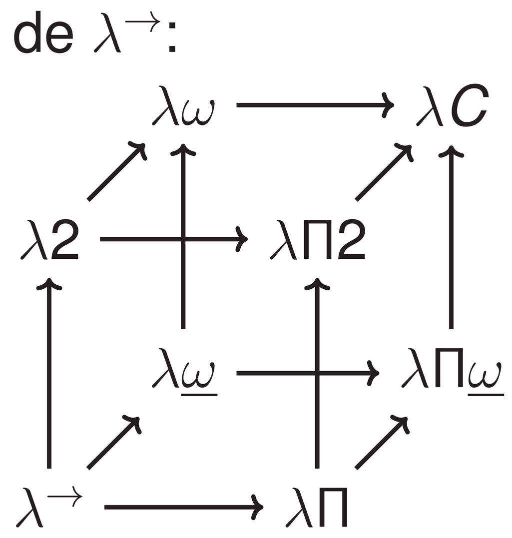
En un sistema así f y g pasan a aplicarse implícitamente (sin intervención de quien programa), y aparecen líneas diagonales que conectan a un programa con los contextos de todos sus correspondientes por isomorfismos.

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.

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 ($\uparrow$)**: tipos como parámetro y cuantificación del tipo, e.g. *length* $: \forall a.[a] \to Int$
- **Constructores de tipos ($\nearrow$)**: funciones en el nivel de los tipos, e.g. *Maybe* $: * \to *$



- **Tipos dependientes ($\to$)**: términos como parámetro de los tipos, e.g. vectores cuyo tamaño se expresa en el tipo: $[1, 1, 1]$: *Vec 3*

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.

DEPARTAMENTO DE COMPUTACIÓN
Facultad de Ciencias Exactas y Naturales - UBA
ICC Instituto de Ciencias de la Computación
EXACTAS UBA
Más información