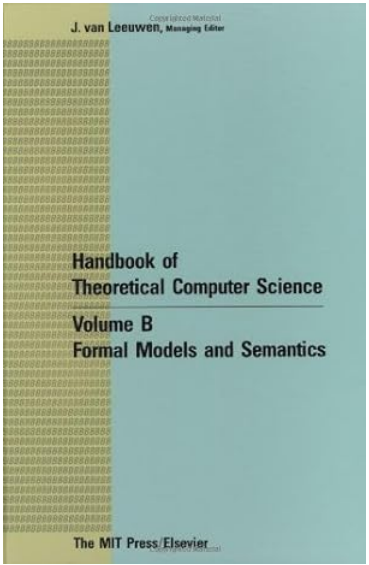
SI1001 Teoría de la Computación
Lambda cálculo

Andrés Sicard Ramírez

Universidad EAFIT

Semestre 2025-1

Texto guía



CHAPTER 7

Functional Programming and Lambda Calculus

H. P. BARENDREGT

Faculty of Mathematics and Computer Science, Catholic University Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, Netherlands

Contents

Preliminares

Convenciones

- La numeración (de secciones, definiciones, teoremas, figuras, páginas, etc.) en estas diapositivas corresponde a la numeración en [Bar1992a].
- Los números naturales incluyen el cero, es decir, $\mathbb{N} = \{0, 1, 2, \dots\}$.
- El conjunto potencia de un conjunto A es denotado $\mathcal{P}A$.
- Los términos «definición inductiva» y «definición recursiva» se usan como sinónimos.
- En las definiciones inductivas se sobreentiende que el conjunto definido es el menor conjunto que satisface las condiciones.

Agradecimientos

Agradezco a mi colega Sergio Steven Ramírez Rico por señalarme algunas correcciones en una versión anterior de estas diapositivas.

Esquema de la presentación

Tema

- Introducción
- Descripción formal del lambda cálculo
- El lambda cálculo como una teoría formal
- Combinadores
- Combinadores de punto fijo
- Codificación de Church
- Codificación de Barendregt
- Representación de las funciones computables
- Referencias

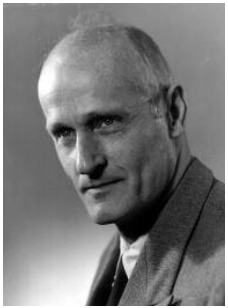
Introducción

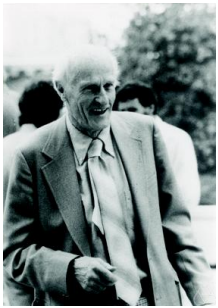
Alonzo Church (1903 – 1995)[a]

[a] Figuras tomadas de History of computers, Wikipedia y MacTutor History of Mathematics.

Introducción

Stephen Cole Kleene (1909 – 1994)[a]





John Barkley Rosser (1907 – 1989)[b]

[a] Figuras tomadas de MacTutor History of Mathematics y Oberwolfach.
[b] Figura tomada de la John Simon Guggenheim Memorial Foundation.

Introducción

- Un sistema formal creado por Alonzo Church alrededor de 1930 y sus estudiantes Stephen Kleene y J. Barkley Rosser.
- El objetivo inicial fue usar el lambda cálculo para las fundaciones de la matemáticas.
- Empleado para estudiar funciones y recursividad.
- Modelo de computabilidad.
- Base de los lenguajes de programación funcionales (Lisp, Haskell, ML, Scheme, etc.)
- Notación (*p. ej.* funciones anónimas y curryficación).

Bibliografía

- Artículos introductorios: [Bar1992b, § 2] y [BB2000].
- Una presentación sucinta de la teoría: [Sel2009, Apéndice A].
- Una presentación de los desarrollos actuales: [CH2009]
- Un artículo acerca de la historia del lambda cálculo: [Ros1984]. Véase también [CH2009] y [Sel2009].
- Libro de texto: [HS2008].
- La «biblia»: [Bar2004].

Aplicación, λ-abstracción y β-conversión (informalmente)

En el tablero.

Introducción

Definición

Una función es de **orden superior** sii

(i) recibe (al menos) una función como un argumento o

(ii) retorna una función como resultado.

Introducción

Definición

Una función es de **orden superior** sii

(i) recibe (al menos) una función como un argumento o

(ii) retorna una función como resultado.

Ejemplos

En el tablero.

Introducción

Definición

Una función es de **orden superior** sii

(i) recibe (al menos) una función como un argumento o

(ii) retorna una función como resultado.

Ejemplos

En el tablero.

Observación

Las funciones de orden superior son usualmente llamadas «funcionales» en matemáticas.

Funciones de varias variables

Funciones de varias variables pueden ser representadas por funciones de orden superior y usos repetidos de aplicación. Esta técnica fue propuesta por Schönfinkel, pero es llamada **curryficación** debido a que fue popularizada por Haskell Curry.

Ejemplo (curryficación)

Sea $g : X \times Y \to Z$ una función de dos variables. Podemos definir dos funciones f_x y f por:

$$f_x : Y \to Z \qquad\qquad f : X \to (Y \to Z)$$
$$f_x = \lambda y.g(x, y), \qquad\qquad f = \lambda x.f_x.$$

Entonces $(f\,x)\,y = f_x\,y = g(x, y)$. Es decir, la función de dos variables

$$g : X \times Y \to Z$$

es representada por la función curryficada de orden superior.

$$f : X \to (Y \to Z).$$

Tema

- Introducción
- Descripción formal del lambda cálculo
- El lambda cálculo como una teoría formal
- Combinadores
- Combinadores de punto fijo
- Codificación de Church
- Codificación de Barendregt
- Representación de las funciones computables
- Referencias

Lambda términos

Definición 2.1.1

Sea V un conjunto enumerable de variables. El conjunto de **lambda términos** o λ**-términos**, denotado por Λ, es inductivamente definido por:

$$x \in V \Rightarrow x \in \Lambda \qquad \text{(variable)}$$

$$M, N \in \Lambda \Rightarrow (M\ N) \in \Lambda \qquad \text{(aplicación)}$$

$$M \in \Lambda, x \in V \Rightarrow (\lambda x.M) \in \Lambda \qquad (\lambda\text{-abstracción})$$

Lambda términos

Definición 2.1.1

Sea V un conjunto enumerable de variables. El conjunto de **lambda términos** o λ-**términos**, denotado por Λ, es inductivamente definido por:[a]

$$x \in V \Rightarrow x \in \Lambda \qquad \text{(variable)}$$

$$M, N \in \Lambda \Rightarrow (M\ N) \in \Lambda \qquad \text{(aplicación)}$$

$$M \in \Lambda, x \in V \Rightarrow (\lambda x.M) \in \Lambda \qquad (\lambda\text{-abstracción})$$

[a] La definición en el texto guía incluye constantes las cuales no serán usadas en el curso.

Lambda términos

Ejemplos

En el tablero.

Lambda términos

Definición

EL conjunto de λ-términos Λ también es definido frecuentemente empleando reglas de inferencia. Sea V un conjunto enumerable de variables, entonces:

$$\frac{x \in V}{x \in \Lambda} \text{ (variable)}$$

$$\frac{M \in \Lambda \qquad N \in \Lambda}{(M\,N) \in \Lambda} \text{ (aplicación)}$$

$$\frac{M \in \Lambda \qquad x \in V}{(\lambda x.M) \in \Lambda} \text{ (λ-abstracción)}$$

Lambda términos

Observación

Usualmente, el conjunto de λ-términos Λ es introducido por una gramática abstracta.[a]

$$
\begin{aligned}
\Lambda \ni t ::= {} & x && \text{(variable)} \\
| \; & t\, t && \text{(aplicación)} \\
| \; & \lambda x.t && (\lambda\text{-abstracción})
\end{aligned}
$$

[a] Véase, *p. ej.* [Pie2002].

Lambda términos

Convenciones

- Las variables serán denotadas por letras minúsculas $x, y, z, \ldots$.
- Los λ-términos serán denotados por $L, M, N, \ldots$.

Notación

La expresión $M \equiv N$ denota que M y N son el mismo λ-término (identidad sintáctica).

Ligadura de variables

Definición

La ocurrencia de una variable x en un λ-término M es **libre** sii x no está en el alcance de λx. En caso contrario, la ocurrencia de la variable x está **ligada** por la abstracción λx.

Ligadura de variables

Definición 2.1.4.(i)

El **conjunto de variables libres** de un λ-término M, denotado por $\mathsf{FV}(M)$, es inductivamente definido por:

$$\begin{aligned}
&\mathsf{FV} : \Lambda \to \mathcal{P}V \\
&\mathsf{FV}(x) &&:= \{x\}, \\
&\mathsf{FV}(M\,N) &&:= \mathsf{FV}(M) \cup \mathsf{FV}(N), \\
&\mathsf{FV}(\lambda x.M) &&:= \mathsf{FV}(M) - \{x\}.
\end{aligned}$$

Sustitución

Definición

El resultado de **sustituir** cada ocurrencia libre de una variable x por un λ-término L en un λ-término M, cambiando variables libres para evitar su captura, denotado por $M[\,x \mapsto L\,]$, está definido por [HS2008, Definition 1.12]:

$$
\begin{aligned}
x[\,x \mapsto L\,] &:= L; && \\
y[\,x \mapsto L\,] &:= y, && \text{si } y \not\equiv x; \\
(M\,Q)[\,x \mapsto L\,] &:= (M[\,x \mapsto L\,])\,(Q[\,x \mapsto L\,]); && \\
(\lambda x.M)[\,x \mapsto L\,] &:= \lambda x.M; && \\
(\lambda y.M)[\,x \mapsto L\,] &:= \lambda y.M, && \text{si } y \not\equiv x \text{ y } x \notin \mathsf{FV}(M); \\
(\lambda y.M)[\,x \mapsto L\,] &:= \lambda y.M[\,x \mapsto L\,], && \text{si } y \not\equiv x, x \in \mathsf{FV}(M) \text{ y } y \notin \mathsf{FV}(L); \\
(\lambda y.M)[\,x \mapsto L\,] &:= \lambda z.(M[\,y \mapsto z\,])[\,x \mapsto L\,], && \text{si } y \not\equiv x, x \in \mathsf{FV}(M) \text{ y } y \in \mathsf{FV}(L);
\end{aligned}
$$

donde en la última ecuación, la variable z es seleccionada tal que $z \notin \mathsf{FV}(L\,M)$.

Sustitución

Ejemplos

En el tablero.

Lambda términos

Convenciones

- Los paréntesis más externos no son escritos.
- La aplicación tiene mayor precedencia que la λ-abstracción, es decir,

$$\lambda x.\, M\, N := (\lambda x.\, (M\, N)).$$

- La aplicación asocia a la izquierda, es decir,

$$M\, N_1\, N_2 \ldots N_k := (\ldots ((M\, N_1)\, N_2) \ldots N_k).$$

- La λ-abstracción asocia a la derecha, es decir,

$$\lambda x_1.\, \lambda x_2. \ldots \lambda x_n.\, M := (\lambda x_1.\, (\lambda x_2.\, (\ldots (\lambda x_n.M) \ldots))).$$

Convenciones

Ejemplo

$((((\lambda\, x.\, (\lambda\, y.\, (\lambda\, z.\, ((x\, z)\, (y\, z)))))\, u)\, v)\, w)$

$\equiv$ (paréntesis externos son removidos)

$(((\lambda\, x.\, (\lambda\, y.\, (\lambda\, z.\, ((x\, z)\, (y\, z)))))\, u)\, v)\, w$

$\equiv$ (λ-abstracción asocia a la derecha)

$(((\lambda\, x.\, \lambda\, y.\, (\lambda\, z.\, ((x\, z)\, (y\, z))))\, u)\, v)\, w$

$\equiv$ (λ-abstracción asocia a la derecha)

$(((\lambda\, x.\, \lambda\, y.\, \lambda\, z.\, ((x\, z)\, (y\, z)))\, u)\, v)\, w$

$\equiv$ (aplicación mayor prec. que λ-abst.)

$(((\lambda\, x.\, \lambda\, y.\, \lambda\, z.\, (x\, z)\, (y\, z))\, u)\, v)\, w$

$\equiv$ (aplicación asocia a la izquierda)

$((\lambda\, x.\, \lambda\, y.\, \lambda\, z.\, (x\, z)\, (y\, z))\, u)\, v\, w$

$\equiv$ (aplicación asocia a la izquierda)

$(\lambda\, x.\, \lambda\, y.\, \lambda\, z.\, (x\, z)\, (y\, z))\, u\, v\, w$

$\equiv$ (aplicación asocia a la izquierda)

$(\lambda\, x.\, \lambda\, y.\, \lambda\, z.\, x\, z\, (y\, z))\, u\, v\, w$

Tema

La teoría $\boldsymbol{\lambda}$

Introducción

- *«The principal object of study in the λ-calculus is the set of λ-terms modulo convertibility.»* [Bar2004, pág. 22].
- La relación de convertibilidad, denotada $=_{\mathrm{conv}}$, es definida por la teoría $\boldsymbol{\lambda}$.

La teoría $\boldsymbol{\lambda}$

Definición

Los **términos** de la teoría $\boldsymbol{\lambda}$ son los λ-términos.

Definición

Las **fórmulas** de la teoría $\boldsymbol{\lambda}$ son de la forma $M =_{\mathrm{conv}} N$, donde M, N son λ-términos.

Definición 2.1.5

Sean L, M, N tres λ-términos y sea x una variable. La relación de convertibilidad $=_{\text{conv}}$ es definida inductivamente por:

- Axioma β
  $(\lambda x.M)\, N =_{\text{conv}} M[\, x \mapsto N\,]$.
- Axiomas y reglas de inferencia para ser una relación de equivalencia
  (i) $M =_{\text{conv}} M$ (refl),
  (ii) $M =_{\text{conv}} N \Rightarrow N =_{\text{conv}} M$ (sim),
  (iii) $M =_{\text{conv}} N, N =_{\text{conv}} L \Rightarrow M =_{\text{conv}} L$ (trans).
- Reglas de inferencia de compatibilidad
  (i) $M =_{\text{conv}} N \Rightarrow M\, L =_{\text{conv}} N\, L$ (appd),
  (ii) $M =_{\text{conv}} N \Rightarrow L\, M =_{\text{conv}} L\, N$ (appi),
  (iii) $M =_{\text{conv}} N \Rightarrow \lambda x.M =_{\text{conv}} \lambda x.N$ (abst).

La teoría λ

Definición

Sean L, M, N tres λ-términos y sea x una variable. La relación de convertibilidad $=_{\text{conv}}$ también se puede definir empleando reglas de inferencia:

- Axioma β

$$\frac{}{(\lambda x.M)\, N =_{\text{conv}} M[\, x \mapsto N\,]} \text{ (axioma } \beta\text{)}$$

- Axiomas y reglas de inferencia para ser una relación de equivalencia

$$\frac{}{M =_{\text{conv}} M} \text{ (refl)} \qquad \frac{M =_{\text{conv}} N}{N =_{\text{conv}} M} \text{ (sim)} \qquad \frac{M =_{\text{conv}} N \qquad N =_{\text{conv}} L}{M =_{\text{conv}} L} \text{ (trans)}$$

- Reglas de inferencia de compatibilidad

$$\frac{M =_{\text{conv}} N}{M\, L =_{\text{conv}} N\, L} \text{ (appd)} \qquad \frac{M =_{\text{conv}} N}{L\, M =_{\text{conv}} L\, N} \text{ (appi)} \qquad \frac{M =_{\text{conv}} N}{\lambda x.M =_{\text{conv}} \lambda x.N} \text{ (abst)}$$

La teoría λ

Notación

Si $M =_{\text{conv}} N$ es un teorema, éste es denotado por $\boldsymbol{\lambda} \vdash M =_{\text{conv}} N$.

La teoría $\boldsymbol{\lambda}$

Notación

Si $M =_{\text{conv}} N$ es un teorema, éste es denotado por $\boldsymbol{\lambda} \vdash M =_{\text{conv}} N$.

Definición 2.1.5

Los λ-términos M y N son β-**convertibles** sii $\boldsymbol{\lambda} \vdash M =_{\text{conv}} N$.

La teoría $\boldsymbol{\lambda}$

Notación

Si $M =_{\text{conv}} N$ es un teorema, éste es denotado por $\boldsymbol{\lambda} \vdash M =_{\text{conv}} N$.

Definición 2.1.5

Los λ-términos M y N son β-**convertibles** sii $\boldsymbol{\lambda} \vdash M =_{\text{conv}} N$.

Observación

La teoría $\boldsymbol{\lambda}$ es

- una teoría ecuacional,
- *logic-free*, es decir, las fórmulas de la teoría no contienen constantes lógicas.

Ejemplo

Sean M y N dos λ-términos. Demostrar que $\boldsymbol{\lambda} \vdash (\lambda x.\, \lambda y.\, x)\, M\, N =_{\text{conv}} M$.

(i) Una demostración en forma de árbol

$$\dfrac{\dfrac{\dfrac{}{(\lambda x.\, \lambda y.\, x)\, M =_{\text{conv}} \lambda y.M}\ (\text{axioma } \beta)}{(\lambda x.\, \lambda y.\, x)\, M\, N =_{\text{conv}} (\lambda y.M)\, N}\ (\text{appd}) \qquad \dfrac{}{(\lambda y.M)\, N =_{\text{conv}} M}\ (\text{axioma } \beta)}{(\lambda x.\, \lambda y.\, x)\, M\, N =_{\text{conv}} M}\ (\text{trans})$$

La teoría λ

Ejemplo

Sean M y N dos λ-términos. Demostrar que $\boldsymbol{\lambda} \vdash (\lambda x.\, \lambda y.\, x)\, M\, N =_{\text{conv}} M$.

(i) Una demostración en forma de árbol

$$\dfrac{\dfrac{\dfrac{}{(\lambda x.\, \lambda y.\, x)\, M =_{\text{conv}} \lambda y.M}\ (\text{axioma } \beta)}{(\lambda x.\, \lambda y.\, x)\, M\, N =_{\text{conv}} (\lambda y.M)\, N}\ (\text{appd}) \qquad \dfrac{}{(\lambda y.M)\, N =_{\text{conv}} M}\ (\text{axioma } \beta)}{(\lambda x.\, \lambda y.\, x)\, M\, N =_{\text{conv}} M}\ (\text{trans})$$

(ii) Una demostración lineal

1. $(\lambda x.\, \lambda y.\, x)\, M =_{\text{conv}} \lambda y.M$ (axioma β)
2. $(\lambda x.\, \lambda y.\, x)\, M\, N =_{\text{conv}} (\lambda y.M)\, N$ (appd en 1)
3. $(\lambda y.M)\, N =_{\text{conv}} M$ (axioma β)
4. $(\lambda x.\, \lambda y.\, x)\, M\, N =_{\text{conv}} M$ (trans en 2 y 3)

Observación

Church empleó el nombre «conv» para denotar la relación de convertibilidad (véase, *p. ej.* [Chu1951]). El texto guía usa el símbolo de la igualdad «$=$», tal vez siguiendo a [CF1958, § 3.D.3].

Alfa congruencia

Definición

Un **cambio de variables ligadas** en un λ-término M es reemplazar un subtérmino $\lambda x.N$ de M por $\lambda y.(N[x \mapsto y])$ donde la variable y no ocurre en el λ-término N [Bar2004, Definición 2.1.11].

Definición

Un λ-término M es α-**congruente** con un λ-término N, denotado por $M \equiv_\alpha N$, sii N resulta de M por un número finito (quizás cero) de cambios de variables ligadas [Bar2004, Definición 2.1.11].

Ejemplos

En el tablero.

Alfa congruencia

Teorema

La relación $\equiv_\alpha$ es una relación de equivalencia.[a]

Convención

Nosotros sintácticamente identificamos λ-términos que son α-congruentes— siguiendo [Bar2004, Convención 2.1.12]—, es decir,

$$M \equiv N := M \equiv_\alpha N.$$

[a] Véase, *p. ej.* [HS2008, Lema 1.19b].

Tema

Combinadores

Definición 2.1.4.(ii)

Un **combinador** (o λ-**término cerrado**) es un λ-término sin variables libres.

Combinadores

Ejemplo 2.1.6 (algunos combinadores comunes)

$$\mathsf{I} := \lambda x.x \qquad \text{(identidad)}$$

$$\mathsf{K} := \lambda\, x.\, \lambda\, y.\, x \qquad \text{(proyección primera componente)}$$

$$\mathsf{K}_* := \lambda\, x.\, \lambda\, y.\, y \qquad \text{(proyección segunda componente)}$$

$$\mathsf{S} := \lambda\, f.\, \lambda\, g.\, \lambda\, x.\, f\, x\, (g\, x) \qquad \text{(composición fuerte)}$$

Ejemplo 2.1.6 (algunos combinadores comunes)

$$\begin{aligned}
\mathsf{I} &:= \lambda x.x && \text{(identidad)}\\
\mathsf{K} &:= \lambda\, x.\, \lambda\, y.\, x && \text{(proyección primera componente)}\\
\mathsf{K}_* &:= \lambda\, x.\, \lambda\, y.\, y && \text{(proyección segunda componente)}\\
\mathsf{S} &:= \lambda\, f.\, \lambda\, g.\, \lambda\, x.\, f\, x\, (g\, x) && \text{(composición fuerte)}
\end{aligned}$$

Teorema

Sean L, M, N λ-términos, entonces

$$\begin{aligned}
\boldsymbol{\lambda} \vdash \mathsf{I}\, M &=_{\text{conv}} M,\\
\boldsymbol{\lambda} \vdash \mathsf{K}\, M\, N &=_{\text{conv}} M,\\
\boldsymbol{\lambda} \vdash \mathsf{K}_*\, M\, N &=_{\text{conv}} N,\\
\boldsymbol{\lambda} \vdash \mathsf{S}\, M\, N\, L &=_{\text{conv}} M\, L\, (N\, L).
\end{aligned}$$

Combinadores

Observación

Los programas en un lenguaje de programación basado en el lambda cálculo son combinadores. combinators.

Observación

Los combinadores K y S (es decir, la lógica combinatoria) son un lenguaje Turing-completo.

Tema

Combinadores de punto fijo

Teorema 2.1.7.(i) (teorema de punto fijo)

Para cualquier λ-término F, existe un λ-término X, tal que

$$\boldsymbol{\lambda} \vdash F\, X =_{\mathrm{conv}} X.$$

Combinadores de punto fijo

Teorema 2.1.7.(i) (teorema de punto fijo)

Para cualquier λ-término F, existe un λ-término X, tal que

$$\boldsymbol{\lambda} \vdash F\,X =_{\text{conv}} X.$$

Demostración.

Sea $W := \lambda x.\, F\,(x\,x)$ y sea $X := W\,W$. Entonces,

$$\begin{aligned} X &\equiv W\,W \\ &\equiv (\lambda x.\, F\,(x\,x))\,W \\ &=_{\text{conv}} F\,(W\,W) \\ &\equiv F\,X, \end{aligned}$$

Es decir, $\boldsymbol{\lambda} \vdash F\,X =_{\text{conv}} X$. ■

Combinadores de punto fijo

Definición

Un **combinador de punto fijo** es un combinador fix tal que, para cualquier λ-término F,

$$\boldsymbol{\lambda} \vdash \mathsf{fix}\, F =_{\mathrm{conv}} F\,(\mathsf{fix}\, F).$$

Combinadores de punto fijo

Teorema 2.1.7.(ii)

El combinador $\mathsf{Y} := \lambda f.\, V\, V$, donde $V := \lambda x.\, f\,(x\,x)$, es un combinador de punto fijo.

Combinadores de punto fijo

Teorema 2.1.7.(ii)

El combinador $\mathsf{Y} := \lambda f.\, V\, V$, donde $V := \lambda x.\, f\,(x\,x)$, es un combinador de punto fijo.

Demostración

En el tablero.

Combinadores de punto fijo

Teorema 2.1.7.(ii)

El combinador $\mathsf{Y} := \lambda f.\, V\, V$, donde $V := \lambda x.\, f\,(x\,x)$, es un combinador de punto fijo.[a]

Demostración

En el tablero.

[a] De acuerdo a [HS2008, pág. 36], este combinador fue insinuado por Curry en 1929 y publicado por primera vez por Rosenbloom [Ros1950]. Véase también [Bar2004, Corollary 6.1.3].

Combinadores de punto fijo

Teorema

El combinador $\mathsf{U}\,\mathsf{U}$, donde $\mathsf{U} := \lambda u.\, \lambda x.\, x\,(u\,u\,x)$, es un combinador de punto fijo.[a]

[a] Este combinador fue definido por Turing [Tur1937]. Véase también [Bar2004, Definition 6.1.4].

Recursividad usando combinadores de punto fijo

Descripción

Los combinadores de punto fijo permiten representar la recursividad en el lambda cálculo.

Recursividad usando combinadores de punto fijo

Ejemplo

Un ejemplo informal usando un combinador de punto fijo fix para definir la función factorial [Pey1987, § 2.4.1].

$$\begin{aligned} \mathsf{fac} &:= \lambda n.\, \mathsf{if}\,(n == 0)\,\mathsf{then}\,1\,\mathsf{else}\,n \times \mathsf{fac}\,(n-1) && \text{(combinador)} \\ &\equiv \lambda n.\, (\ldots \mathsf{fac} \ldots) && \text{(combinador recursivo)} \\ &\equiv (\lambda f.\, \lambda n.\, (\ldots f \ldots))\,\mathsf{fac} && (\lambda\text{-abstracción en fac}) \end{aligned}$$

(continua en la próxima diapositiva)

Recursividad usando combinadores de punto fijo

Ejemplo (continuación)

Ahora, sea h el combinador no recursivo

$$\mathsf{h} := \lambda f.\, \lambda n.\, (\ldots f \ldots),$$

y observamos que fac es un punto fijo de h

$$\mathsf{fac} := \mathsf{h}\,\mathsf{fac}.$$

Entonces, podemos definir fac empleando un combinador de punto fijo fix

$$\mathsf{fac} := \mathsf{fix}\,\mathsf{h}.$$

(continua en la próxima diapositiva)

Recursividad usando combinadores de punto fijo

Ejemplo (continuación)

$$
\begin{aligned}
\mathsf{fac}\,1 &\equiv \mathsf{fix}\,\mathsf{h}\,1 \\
&=_{\mathrm{conv}} \mathsf{h}\,(\mathsf{fix}\,\mathsf{h})\,1 \\
&\equiv (\lambda f.\,\lambda n.\,(\ldots f \ldots))\,(\mathsf{fix}\,\mathsf{h})\,1 \\
&=_{\mathrm{conv}} \mathsf{if}\,(1 == 0)\,\mathsf{then}\,1\,\mathsf{else}\,1 \times (\mathsf{fix}\,\mathsf{h}\,0) \\
&=_{\mathrm{conv}} 1 \times (\mathsf{fix}\,\mathsf{h}\,0) \\
&=_{\mathrm{conv}} 1 \times (\mathsf{h}\,(\mathsf{fix}\,\mathsf{h})\,0) \\
&\equiv 1 \times ((\lambda f.\,\lambda n.\,(\ldots f \ldots))\,(\mathsf{fix}\,\mathsf{h})\,0) \\
&=_{\mathrm{conv}} 1 \times (\mathsf{if}\,(0 == 0)\,\mathsf{then}\,1\,\mathsf{else}\,1 * (\mathsf{fix}\,\mathsf{h}\,(-1))) \\
&=_{\mathrm{conv}} 1 \times 1 \\
&=_{\mathrm{conv}} 1
\end{aligned}
$$

Tema

Codificación de Church

Definición 2.1.9.(i)

Sean F y M dos λ-términos y sea $n \in \mathbb{N}$. Inductivamente definimos $F^n(M)$ por:

$$F^0(M) := M,$$
$$F^{n+1}(M) := F(F^n(M)).$$

Codificación de Church

Definición

Los **numerales de Church**, denotados por c_n con $n \in \mathbb{N}$, codifican los números naturales:

$$\mathsf{c}_n := \lambda f.\, \lambda x.\, f^n(x).$$

Codificación de Church

Ejemplo

Algunos numerales.

$$\mathsf{c}_0 \equiv \lambda\, f.\, \lambda\, x.\, x,$$
$$\mathsf{c}_1 \equiv \lambda\, f.\, \lambda\, x.\, f\, x,$$
$$\mathsf{c}_2 \equiv \lambda\, f.\, \lambda\, x.\, f\, (f\, x),$$
$$\mathsf{c}_3 \equiv \lambda\, f.\, \lambda\, x.\, f\, (f\, (f\, x)).$$

Codificación de Church

Sucesor

Un combinador para el sucesor es definido por

$$\mathsf{succ} := \lambda\, n\, f\, x.\, f\, (n\, f\, x),$$

donde para todo $n \in \mathbb{N}$,

$$\boldsymbol{\lambda} \vdash \mathsf{succ}\, \mathsf{c}_n =_{\mathrm{conv}} \mathsf{c}_{n+1}.$$

Codificación de Church

Proposición 2.1.10 (adición, multiplicación y exponenciación)

Sean

$$\begin{aligned} \mathsf{add} &:= \lambda\, x.\, \lambda\, y.\, \lambda\, p.\, \lambda\, q.\, x\, p\, (y\, p\, q), \\ \mathsf{mult} &:= \lambda\, x. \lambda\, y. \lambda\, z.\, x\, (y\, z), \\ \mathsf{exp} &:= \lambda\, x.\, \lambda y.\, y\, x, \end{aligned}$$

entonces para todo $m, n \in \mathbb{N}$,

$$\begin{aligned} \boldsymbol{\lambda} \vdash \mathsf{add}\, \mathsf{c}_m\, \mathsf{c}_n &=_{\mathrm{conv}} \mathsf{c}_{m+n}. \\ \boldsymbol{\lambda} \vdash \mathsf{mult}\, \mathsf{c}_m\, \mathsf{c}_n &=_{\mathrm{conv}} \mathsf{c}_{m \times n}. \\ \boldsymbol{\lambda} \vdash \mathsf{exp}\, \mathsf{c}_m\, \mathsf{c}_n &=_{\mathrm{conv}} \mathsf{c}_{m^n}, \text{ excepto para } n = 0. \end{aligned}$$

Codificación de Church

Definición 2.2.1

Codificamos los booleanos por

$$\mathsf{true} := \mathsf{K}, \qquad \mathsf{false} := \mathsf{K}_*,$$

donde para todos $M, N \in \Lambda$,

$$\boldsymbol{\lambda} \vdash \mathsf{true}\, M\, N =_{\mathrm{conv}} M,$$
$$\boldsymbol{\lambda} \vdash \mathsf{false}\, M\, N =_{\mathrm{conv}} N.$$

Codificación de Church

Test para cero

Un combinador para un test para cero es definido por

$$\mathsf{isZero} := \lambda n.\, n\,(\lambda x.\mathsf{false})\,\mathsf{true},$$

donde para todo $n \in \mathbb{N}$,

$$\begin{aligned} \boldsymbol{\lambda} \vdash \mathsf{isZero}\,\mathsf{c}_0 &=_{\mathrm{conv}} \mathsf{true}, \\ \boldsymbol{\lambda} \vdash \mathsf{isZero}\,\mathsf{c}_{n+1} &=_{\mathrm{conv}} \mathsf{false}. \end{aligned}$$

Codificación de Church

Predecesor

Un combinador para el predecesor es definido por

$$\mathsf{pred} := \lambda\, n.\, \lambda\, f.\, \lambda\, x.\, n\, (\lambda\, g.\, \lambda\, h.\, h\, (g\, f))\, (\lambda u.x)\, (\lambda u.u),$$

donde para todo $n \in \mathbb{N}$,

$$\begin{aligned} \boldsymbol{\lambda} \vdash \mathsf{pred}\, \mathsf{c}_0 &=_{\mathrm{conv}} \mathsf{c}_0, \\ \boldsymbol{\lambda} \vdash \mathsf{pred}\, \mathsf{c}_{n+1} &=_{\mathrm{conv}} \mathsf{c}_n. \end{aligned}$$

Tema

- Introducción
- Descripción formal del lambda cálculo
- El lambda cálculo como una teoría formal
- Combinadores
- Combinadores de punto fijo
- Codificación de Church
- Codificación de Barendregt
- Representación de las funciones computables
- Referencias

Codificación de Barendregt

Definición 2.2.1

Codificamos los booleanos por

$$\mathsf{true} := \mathsf{K}, \qquad \mathsf{false} := \mathsf{K}_*,$$

donde para todos $M, N \in \Lambda$,

$$\boldsymbol{\lambda} \vdash \mathsf{true}\, M\, N =_{\mathrm{conv}} M,$$
$$\boldsymbol{\lambda} \vdash \mathsf{false}\, M\, N =_{\mathrm{conv}} N.$$

Codificación de Barendregt

Definición 2.2.2

Sean M y N dos λ-términos. Codificamos un par ordenado de M y N por:

$$\begin{aligned} [\, M, N \,] &:= \lambda z.\, z\, M\, N, \\ \mathsf{fst} &:= \lambda p.\, p\, \mathsf{true}, \\ \mathsf{snd} &:= \lambda p.\, p\, \mathsf{false}, \end{aligned}$$

donde para todos $M, N \in \Lambda$,

$$\begin{aligned} \boldsymbol{\lambda} \vdash \mathsf{fst}\, [\, M, N \,] &=_{\mathrm{conv}} M, \\ \boldsymbol{\lambda} \vdash \mathsf{snd}\, [\, M, N \,] &=_{\mathrm{conv}} N. \end{aligned}$$

Codificación de Barendregt

Definición 2.2.2

Sean M y N dos λ-términos. Codificamos un par ordenado de M y N por:[a]

$$
\begin{aligned}
[\,M, N\,] &:= \lambda z.\, z\, M\, N, \\
\mathsf{fst} &:= \lambda p.\, p\, \mathsf{true}, \\
\mathsf{snd} &:= \lambda p.\, p\, \mathsf{false},
\end{aligned}
$$

donde para todos $M, N \in \Lambda$,

$$
\begin{aligned}
\boldsymbol{\lambda} \vdash \mathsf{fst}\,[\,M, N\,] &=_{\mathrm{conv}} M, \\
\boldsymbol{\lambda} \vdash \mathsf{snd}\,[\,M, N\,] &=_{\mathrm{conv}} N.
\end{aligned}
$$

[a] El texto guía no usa un nombre para el combinador fst ni para el combinador snd.

Codificación de Barendregt

Definición 2.2.3

Los **numerales de Barendregt**, denotados por $\overline{n}$, con $n \in \mathbb{N}$, codifican los números naturales:

$$\overline{0} := \mathsf{I},$$
$$\overline{n+1} := [\,\mathsf{false}, \overline{n}\,].$$

Codificación de Barendregt

Lema 2.2.4 (sucesor, predecesor y test para cero)

Sean

$$\mathsf{S}^+ := \lambda x.[\,\mathsf{false}, \overline{x}\,], \qquad \mathsf{P}^- := \lambda x.\, x\,\mathsf{false}, \qquad \mathsf{Zero} := \lambda x.\, x\,\mathsf{true},$$

entonces para todo $n \in \mathbb{N}$,

$$\begin{aligned}
\boldsymbol{\lambda} \vdash \mathsf{S}^+\,\overline{n} &=_{\mathrm{conv}} \overline{n+1},\\
\boldsymbol{\lambda} \vdash \mathsf{P}^-\,\overline{0} &=_{\mathrm{conv}} \mathsf{false},\\
\boldsymbol{\lambda} \vdash \mathsf{P}^-\,\overline{n+1} &=_{\mathrm{conv}} \overline{n},\\
\boldsymbol{\lambda} \vdash \mathsf{Zero}\,\overline{0} &=_{\mathrm{conv}} \mathsf{true},\\
\boldsymbol{\lambda} \vdash \mathsf{Zero}\,\overline{n+1} &=_{\mathrm{conv}} \mathsf{false}.
\end{aligned}$$

Tema

Representación de las funciones computables

Definición

Una **función numérico-teórica** es una función

$$\mathbb{N}^p \to \mathbb{N}, \text{ con } p \in \mathbb{N}.$$

Representación de las funciones computables

Definición 2.2.5

Sea $f : \mathbb{N}^p \to \mathbb{N}$ una función numérico-teórica. La función f es λ-**definible respecto a la codificación de Barendregt** sii existe un combinador F, tal que para todo $n_1, \ldots, n_p \in \mathbb{N}$,

$$\boldsymbol{\lambda} \vdash \mathsf{F}\,\overline{n_1}\ldots\overline{n_p} =_{\mathrm{conv}} \overline{f(n_1, \ldots, n_p)}.$$

Representación de las funciones computables

Lema 2.2.9

Las funciones iniciales son λ-definibles (respecto a la codificación de Barendregt).

Lema 2.2.10

Las funciones λ-definibles (respecto a la codificación de Barendregt) son cerradas bajo composición.

Lema 2.2.11

Las funciones λ-definibles (respecto a la codificación de Barendregt) son cerradas bajo recursion primitiva.

Lema 2.2.12

Las funciones λ-definibles (respecto a la codificación de Barendregt) son cerradas bajo minimalización.

Representación de las funciones computables

Teorema 2.2.13

Las funciones recursivas son λ-definibles (respecto a la codificación de Barendregt).

Representación de las funciones computables

Definición 2.2.5

Sea $f : \mathbb{N}^p \to \mathbb{N}$ una función numérico-teórica. La función f es λ-**definible respecto a la codificación de Church** sii existe un combinador F, tal que para todo $n_1, \ldots, n_p \in \mathbb{N}$,

$$\boldsymbol{\lambda} \vdash \mathsf{F}\, \mathsf{c}_{n_1} \ldots \mathsf{c}_{n_p} =_{\mathrm{conv}} \mathsf{c}_{f(n_1,\ldots,n_p)}.$$

Representación de las funciones computables

Teorema 2.2.14

Las funciones recursivas son λ-definibles (respecto a la codificación de Church).

Tema

- Introducción
- Descripción formal del lambda cálculo
- El lambda cálculo como una teoría formal
- Combinadores
- Combinadores de punto fijo
- Codificación de Church
- Codificación de Barendregt
- Representación de las funciones computables
- Referencias

Referencias

[Bar2004] H. P. Barendregt. *The Lambda Calculus. Its Syntax and Semantics*. Revised edition, 6th impression. Vol. 103. Studies in Logic and the Foundations of Mathematics. Elsevier, 2004 (vid. págs. 10, 31, 41, 42, 54, 55).

[Bar1992a] Henk Barendregt. «Functional Programming and Lambda Calculus». En: *Handbook of Theoretical Computer Science. Volume B. Formal Models and Semantics*. Ed. por J. van Leeuwen. Second impression. MIT Press, 1992. Cap. 7. DOI: 10.1016/B978-0-444-88074-1.50012-3 (vid. pág. 3).

[Bar1992b] Henk Barendregt. «Lambda Calculi with Types». En: *Handbook of Logic in Computer Science. Volume 2*. Ed. por S. Abramsky, Dov M. Gabbay y T. S. E. Maibaum. Clarendon Press, 1992, págs. 117-309 (vid. pág. 10).

[BB2000] Henk Barendregt y Erik Barendsen. «Introduction to Lambda Calculus». Revisited edition. 2000 (vid. pág. 10).

[CH2009] Felice Cardone y J. Roger Hindley. «Lambda-Calculus and Combinators in the 20th Century». En: *Handbook of Logic in Computer Science*. Ed. por Dov M. Gabbay y John Woods. Vol. 5. Elsevier, 2009, págs. 723-817 (vid. pág. 10).

Referencias

[Chu1951] Alonzo Church. *The Calculi of Lambda-Conversion*. Second printing. Princenton University Press, 1951 (vid. pág. 40).

[CF1958] Haskell B. Curry y Robert Feys. *Combinatory Logic*. Vol. I. North-Holland Publishing Company, 1958 (vid. pág. 40).

[GW2009] Dov M. Gabbay y John Woods, eds. *Handbook of the History of Logic*. Vol. 5. Elsevier, 2009.

[HS2008] J. Roger Hindley y Jonathan P. Seldin. *Lambda-Calculus and Combinators. An Introduction.* Cambridge University Press, 2008 (vid. págs. 10, 26, 42, 54).

[Pey1987] Simon L. Peyton Jones. *The Implementation of Functional Programming Languages*. Series in Computer Sciences. Prentice-Hall International, 1987 (vid. pág. 57).

[Pie2002] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002 (vid. pág. 22).

[Ros1950] Paul C. Rosenbloom. *The Elements of Mathematical Logic*. Dover Publications, 1950 (vid. pág. 54).

Referencias

[Ros1984] J. Barkley Rosser. «Highlights of the History of Lambda-Calculus». En: *Annals of the History of Computing* 6.4 (1984), págs. 337-349. DOI: 10.1109/MAHC.1984.10040 (vid. pág. 10).

[Sel2009] Jonathan P. Seldin. «The Logic of Church and Curry». En: *Handbook of Logic in Computer Science*. Ed. por Dov M. Gabbay y John Woods. Vol. 5. Elsevier, 2009, págs. 819-873 (vid. pág. 10).

[Tur1937] A. M. Turing. «The $\mathfrak{p}$-Function in λ-K-Conversion». En: *The Journal of Symbolic Logic* 4.2 (1937), pág. 164. DOI: 10.2307/2268281 (vid. pág. 55).