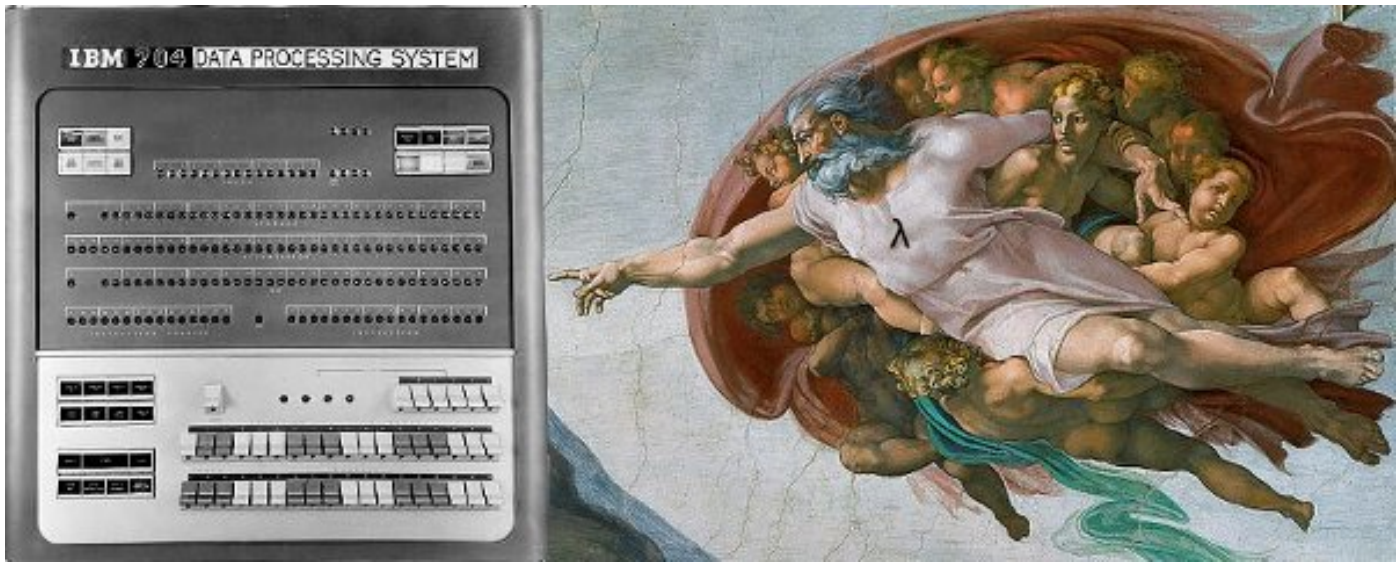
Lambda-Cálculo

Mauro Jaskelioff

6/9/2011



Origen del λ-cálculo

- El λ-cálculo fue inventado por Alonzo Church en la década de 1930.
- Originalmente fue inventado como parte de un sistema formal para modelar la mátematica.
  - ¡Pero es inconsistente!
- Es utilizado para estudiar la computabilidad.
  - En paralelo, Turing presenta su máquina.
- En los 1960s, Peter Landin muestra que se puede usar para dar semántica a los lenguajes de programación (imperativos).
- Los lenguajes funcionales están basados en el λ-cálculo.

SINTAXIS

Sintaxis

- Suponemos la existencia de un conjunto infinito de identificadores
  - x, y ,z, $\ldots$, $x0$, $x1$ denotan elementos de X
- El conjunto Λ de λ-términos se define inductivamente por las siguientes reglas:

$$\frac{x \in X}{x \in \Lambda} \qquad \frac{t \in \Lambda \quad u \in \Lambda}{(t\ u) \in \Lambda} \qquad \frac{x \in X \quad t \in \Lambda}{(\lambda x.t) \in \Lambda}$$

- Ejemplos:
  $x \quad (x\ y) \quad (\lambda x.x) \quad (\lambda x.(\lambda y.((x\ y)\ y)))$

¿Esto es todo?

- Con este pequeño lenguaje se pueden representar todas las funciones computables!
  - (Tesis de Church)
- Esta simpleza hace que:
  - Se facilite la prueba de propiedades.
  - Se use para dar semántica a lenguajes imperativos y funcionales.
  - Su use como metalenguaje para definir otras teorías y cálculos.
- La elegancia hace que sea más práctico!

Convenciones

- Las mayúsculas indican λ-términos arbitrarios (ej: M,N,P)
- Escribimos:

$$\begin{array}{rcl} M\ N\ P & \text{en lugar de} & ((M\ N)\ P) \\ \lambda x.P\ Q & \text{en lugar de} & (\lambda x.P\ Q) \\ \lambda x_1\, x_2 \ldots x_n.M & \text{en lugar de} & (\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.M)\ldots)) \end{array}$$

Ejercicio

Insertar todos los paréntesis y λs en los sig. términos abreviados:

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

Ocurrencias

- La *identidad sintáctica* se denota con $\equiv$
  - $M \equiv N$ iff M es exactamente el mismo término que N.

Definición (Ocurrencia)

La relación P ocurre en Q (o P es un subtérmino de Q) se define inductivamente sobre la estructura de Q

- *P ocurre en P;*
- *si P ocurre en M o en N, entonces P ocurre en $(M\ N)$;*
- *si P ocurre en M o $P \equiv x$, entonces P ocurre en $(\lambda x.M)$.*

Ejercicio

Encontrar las ocurrencias de $(x\ y)$ en los términos
$(\lambda x\ y.x\ y)$ $\quad$ $(z\ (x\ y)\ (\lambda x.y\ (x\ y))\ x\ y)$

Variables libre y ligadas

- para una ocurrencia de $\lambda x.M$ en P, M es el *alcance* de la abstracción λx.
- Hay 3 tipos de ocurrencia de una variable x en un término P
  1. ocurrencia de ligadura (si es la x en un λx)
  2. ocurrencia ligada (si es una x en el alcance de un λx en P).
  3. ocurrencia libre (en cualquier otro caso).
- Llamamos $FV(P)$ al conjunto de las variables libres en P.
- Un *término cerrado* es un término sin variables libres.

Ejemplos

$$(\lambda x.x\ y) \qquad (\lambda x.x\ (\lambda x.x))\ x$$

- Observamos que
  - una misma variable puede ocurrir libre y ligada
  - distintas ocurrencias pueden ligarse a distintas ocurrencias de ligadura
  - la ligadura depende de toda la expresión
    (una ocurrencia cambia de "status" de una subexpresión a la expresión final; ej: x vs. $(\lambda x.x)$)

Ejercicio

Dar las variables libres y las ligaduras y sus alcances en el término

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

Substitución

Definición (Substitución)

Para todo M, N, x se define $M[N/x]$ como el resultado de substituir N por toda ocurrencia libre de x en M. Más precisamente, por inducción sobre la estructura de M.

$$
\begin{array}{lll}
x[N/x] & \equiv N & \\
a[N/x] & \equiv a & (a \not\equiv x) \\
(P\ Q)[N/x] & \equiv (P[N/x]\ Q[N/x]) & \\
(\lambda x.P)[N/x] & \equiv \lambda x.P & \\
(\lambda y.P)[N/x] & \equiv \lambda y.P & \textbf{if } x \notin FV(P) \wedge y \not\equiv x \\
(\lambda y.P)[N/x] & \equiv \lambda y.P[N/x] & \textbf{if } x \in FV(P) \wedge y \notin FV(N) \\
(\lambda y.P)[N/x] & \equiv \lambda z.(P[z/y])[N/x] & \textbf{if } x \in FV(P) \wedge y \in FV(N)
\end{array}
$$

Asumimos que $y \not\equiv x$ y que z es la 1er variable $\notin FV(N\ P)$

α-equivalencia

- Dado una ocurrencia de $\lambda x.M$ en un término P, si y no ocurre en M podemos reemplazar $\lambda x.M$ por:

$$\lambda y.(M[y/x])$$

- Esta operación se llama *cambio de variable ligada* o α-*conversión*.
- Si P puede cambiarse a Q por una serie finita de cambios de variable ligada decimos que P es *congruente* con Q, o que P α-convierte a Q, o

$$P \equiv_\alpha Q$$

- Ejemplo: $\lambda x\ y.x\ (x\ y) \equiv_\alpha \lambda u\ v.u\ (u\ v)$ (Probarlo!)

Propiedades de la α-conversión

Lema

a) *Si* $P \equiv_\alpha Q$ *entonces* $FV(P) = FV(Q)$

b) *La relación* $\equiv_\alpha$ *es una relación de equivalencia, o sea:*

es reflexiva	$P \equiv_\alpha P$
es simétrica	$P \equiv_\alpha Q \Rightarrow Q \equiv_\alpha P$
es transitiva	$P \equiv_\alpha Q \wedge Q \equiv_\alpha R \Rightarrow P \equiv_\alpha R$

c) $M \equiv_\alpha M' \wedge N \equiv_\alpha N' \Rightarrow M[N/x] \equiv_\alpha M'[N'/x]$

- Salvo que se aclare lo contrario, escribiremos simplemente $\equiv$ en lugar de $\equiv_\alpha$.
  - Rara vez nos interesa diferenciar términos α-equivalentes.

SEMÁNTICA

β-reducción

- ¿Cómo calcular con el λ-cálculo?
- Un término $(\lambda x.M)\ N$ representa un operador $(\lambda x.M)$ aplicado a un argumento N.
- El "resultado" se obtiene usando la substitución $M[N/x]$.

Definición (redex, contracción, $\to_\beta$, $\to_\beta^*$)

Un término $(\lambda x.M)\ N$ es un β-redex y $M[N/x]$ su contracción. Si al reemplazar un β-redex en un término P por su contracción obtenemos un término P', decimos que P se β-contrae a P' y escribimos

$$P \to_\beta P'$$

Escribimos $\to_\beta^*$ para la clausura reflexiva-transitiva de $\to_\beta$ y decimos que P β-reduce a Q iff $P \to_\beta^* Q$.

Ejemplos

$$
\begin{array}{lll}
(\lambda x.x\ (x\ y))\ N & \to_\beta & N\ (N\ y) \\
(\lambda x.y)\ N & \to_\beta & y \\
(\lambda x.(\lambda y.y\ x)\ z)\ v & \to_\beta & ((\lambda y.y\ x)\ z)[v/x] \equiv (\lambda y.y\ v)\ z \\
(\lambda x.x\ x)\ (\lambda x.x\ x) & \to_\beta & (x\ x)[(\lambda x.x\ x)/x] \equiv (\lambda x.x\ x)\ (\lambda x.x\ x) \\
 & \to_\beta & (x\ x)[(\lambda x.x\ x)/x] \equiv (\lambda x.x\ x)\ (\lambda x.x\ x) \\
 & \to_\beta & \dots \\
(\lambda x.x\ x\ y)\ (\lambda x.x\ x\ y) & \to_\beta & (\lambda x.x\ x\ y)\ (\lambda x.x\ x\ y)\ y \\
 & \to_\beta & (\lambda x.x\ x\ y)\ (\lambda x.x\ x\ y)\ y\ y \\
 & \to_\beta & \dots
\end{array}
$$

- En los dos últimos ejemplos la reducción es infinita!

Forma Normal β

Definición (Formal Normal β)

Una forma normal β o β-nf es un término que no contiene β-redexes.

- Si un término P β-reduce a una β-nf Q decimos que Q es una forma normal β de P

Ejercicio

Reducir los siguientes términos a β-nf.

$$(\lambda x.x\ y)\ (\lambda u.v\ u\ u) \qquad (\lambda x.x\ x\ y)\ (\lambda y.y\ z)$$

Propiedades de $\rightarrow_\beta^*$

- Nada nuevo es introducido en una reducción.

Lema

$$P \rightarrow_\beta^* Q \quad \Rightarrow \quad FV(P) \supseteq FV(Q)$$

- La relación $\rightarrow_\beta^*$ es preservada por la substitución

Lema

$$P \rightarrow_\beta^* P' \quad \wedge \quad Q \rightarrow_\beta^* Q' \quad \Rightarrow \quad Q[P/x] \rightarrow_\beta^* Q'[P'/x]$$

Confluencia

- Algunos términos tienen más de una reducción

$$(\lambda x.(\lambda y.y\ x)\ z)\ v \ \rightarrow_\beta \ (\lambda y.y\ v)\ z \ \rightarrow_\beta \ z\ v$$
$$(\lambda x.(\lambda y.y\ x)\ z)\ v \ \rightarrow_\beta \ (\lambda x.z\ x)\ v \ \rightarrow_\beta \ z\ v$$

- ¿Reducen siempre a la misma forma normal?

Teorema (Church-Rosser para $\rightarrow_\beta$)

Si $P \rightarrow_\beta M$ y $P \rightarrow_\beta N$, entonces existe T tal que

$$M \rightarrow_\beta T \qquad N \rightarrow_\beta T$$

Corolario

Si P tiene β-nf, ésta es única (módulo $\equiv_\alpha$).

β-equivalencia

Definición (β-equivalencia)

P es β-equivalente a Q (escribimos $P =_\beta Q$) iff Q puede ser obtenido partiendo de P y realizando una serie finita de β-contracciones, β-expansiones (β-contracciones inversas) y α-conversiones.

Lema (Substitución y $=_\beta$)

$$M =_\beta M' \quad \wedge \quad N =_\beta N' \quad \Rightarrow \quad M[N/x] =_\beta M'[N'/x]$$

Teorema (Church-Rosser para $=_\beta$)

Si $P =_\beta Q$ entonces existe T tal que

$$P \to_\beta^* T \quad \wedge \quad Q \to_\beta^* T$$

Extensionalidad

- Las λ-abstracciones representan funciones.
- Sin embargo, $\lambda x.f\ x \neq_\beta f$
- Para tener un cálculo extensional, agregamos una nuevo redex (η-redex)

$$\lambda x.f\ x \to_\eta f$$

- $P \to_{\beta\eta} P' \quad \Leftrightarrow \quad P \to_\beta P'$ o $P \to_\eta P'$
- En forma análoga al caso de β se obtiene $\to^*_{\beta\eta}$, forma normal $\beta\eta$ y equivalencia $=_{\beta\eta}$
- El cálculo $\lambda\beta\eta$ es confluente (hay un teorema de Church-Rosser para $\beta\eta$).

Estrategias de reducción

- Por Church-Rosser si un término tiene una forma normal, ésta es única (probarlo!)
- Ya vimos que $\Omega \equiv (\lambda x.x\ x)\ (\lambda x.x\ x)$ tiene infinitas contracciones.
- Por lo tanto $P \equiv (\lambda x\ y\ .\ y)\ \Omega$ también.
- Sin embargo P tiene una forma normal $(\lambda y.y)$.
  - Claramente, la elección del redex a contraer es importante.
- ¿Cómo pruebo qué un término no tiene forma normal?
- ¿Cómo puedo asegurarme de encontrar la forma normal? (si esta existe)

Reducción Normal

- Un redex es *maximal* si no está contenido en algún otro redex.
- Un redex es *maximal izquierdo* si es el redex maximal de más a la izquierda.
- La estrategia de reducción *normal* es elegir siempre el redex maximal izquierdo.

Teorema

Si la reducción normal de un término X es infinita, X no tiene forma normal.

- Para probar que un término no tiene forma normal basta probarla para la reducción normal
- Si una forma normal existe, la estrategia de reducción normal la encontrará.

PROGRAMACIÓN

Programando con el λ-cálculo

- ¿Cómo escribir un programa en λ-cálculo?
- Necesitamos representar algunos tipos de datos básicos (como naturales, booleanos, etc) con λ-expresiones.
  - Establecemos expresiones que representan los valores del tipo
  - Establecemos expresiones que operan sobre el tipo.
- Nos quedamos satisfechos cuando los valores y operadores del tipo cumplen con una especificación dada.
  - Como el λ-cálculo no tiene tipos, expresiones como $(\mathit{not}\ \underline{2})$ son válidas, pero no nos interesa como se comporten.
  - Escribiremos "definiciones" como $\mathit{True} \equiv (\lambda x\ y.x)$, pero esto es simplemente una abreviación expresada en nuestra metalenguaje.

Booleanos

- Queremos representar los valores *True* y *False*, y la operación *ifthenelse*
- Nuestra especificación es

$$\begin{aligned} &\mathit{ifthenelse}\ \mathit{True}\ \ P\ Q =_\beta P \\ &\mathit{ifthenelse}\ \mathit{False}\ P\ Q =_\beta Q \end{aligned}$$

- Por lo tanto

$$\begin{aligned} &\mathit{ifthenelse}\ \mathit{True} =_\beta \lambda p\ q.p \\ &\mathit{ifthenelse}\ \mathit{False} =_\beta \lambda p\ q.q \end{aligned}$$

- Una solución:

$$\begin{aligned} \mathit{True} &\equiv \lambda p\ q.p \\ \mathit{False} &\equiv \lambda p\ q.q \\ \mathit{ifthenelse} &\equiv \lambda x.x \end{aligned}$$

Mas operaciones sobre Booleanos

- Dados *True*, *False* e *ifthenelse* podemos definir otras funciones (escribimos *ifthenelse* P Q R como **if** P **then** Q **else** R)

$$
\begin{aligned}
\mathit{not} &\equiv \lambda x.\textbf{if } x \textbf{ then } \mathit{False} \textbf{ else } \mathit{True} \\
&\equiv \lambda x.\mathit{ifthenelse}\ x\ \mathit{False}\quad\ \mathit{True} \\
&\equiv \lambda x.(\lambda x.x)\quad x\ (\lambda p\ q.q)\ (\lambda p\ q.p) \\
&\to_\beta \lambda x.x\ (\lambda p\ q.q)\ (\lambda p\ q.p)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{not}\ \mathit{True} &\to_\beta (\lambda x.x\ (\lambda p\ q.q)\ (\lambda p\ q.p))\ (\lambda p\ q.p) \\
&\to_\beta (\lambda p\ q.p)\ (\lambda p\ q.q)\ (\lambda p\ q.p) \\
&\to_\beta (\lambda p\ q.q) \\
&\equiv \mathit{False}
\end{aligned}
$$

$$\mathit{not}\ \mathit{False} \to_\beta \ldots \qquad (\mathit{Ejercicio}!)$$

- Otras funciones:

$$
\begin{aligned}
\mathit{and} &\equiv \lambda x\ y.\textbf{if } x \textbf{ then } y \textbf{ else } \mathit{False} \\
\mathit{or} &\equiv \lambda x\ y.\textbf{if } x \textbf{ then } \mathit{True} \textbf{ else } y
\end{aligned}
$$

Pares

- Queremos representar *pair* y las operaciones *fst* y *snd*
- Nuestra especificación es

$$\begin{aligned} &\mathit{fst}\ (\mathit{pair}\ P\ Q) =_\beta P \\ &\mathit{snd}\ (\mathit{pair}\ P\ Q) =_\beta Q \end{aligned}$$

- Una solución:

$$\begin{aligned} \mathit{pair} &\equiv \lambda x\ y.\lambda b.\textbf{if}\ b\ \textbf{then}\ x\ \textbf{else}\ y \\ \mathit{fst} &\equiv \lambda p.p\ \mathit{True} \\ \mathit{snd} &\equiv \lambda p.p\ \mathit{False} \end{aligned}$$

- Verifiquemos que $\mathit{fst}\ (\mathit{pair}\ x\ y) =_\beta x$:

$$\begin{aligned} \mathit{fst}\ (\mathit{pair}\ x\ y) &=_\beta (\lambda p.p\ \mathit{True})\ (\mathit{pair}\ x\ y) \\ &=_\beta (\mathit{pair}\ x\ y)\ \mathit{True} \equiv (\lambda b.\textbf{if}\ b\ \textbf{then}\ x\ \textbf{else}\ y)\ \mathit{True} \\ &=_\beta \textbf{if}\ \mathit{True}\ \textbf{then}\ x\ \textbf{else}\ y \\ &=_\beta x \end{aligned}$$

Notación (metalenguaje)

- En nuestro metalenguaje anotaremos

$$\begin{array}{ll} F^0 & M \equiv M \\ F^{n+1} & M \equiv F^n\,(FM) \end{array}$$

- Ejemplo:

$$\begin{aligned} (\lambda x\ y.x)^3\ z &\equiv (\lambda x\ y.x)^2\ ((\lambda x\ y.x)\ z) \\ &\equiv (\lambda x\ y.x)^1\ ((\lambda x\ y.x)\ ((\lambda x\ y.x)\ z)) \\ &\equiv (\lambda x\ y.x)^0\ ((\lambda x\ y.x)\ ((\lambda x\ y.x)\ ((\lambda x\ y.x)\ z))) \\ &\equiv (\lambda x\ y.x)\ ((\lambda x\ y.x)\ ((\lambda x\ y.x)\ z)) \end{aligned}$$

- Notar que la notación sólo tiene sentido como parte del metalenguaje.

Numerales de Church

Definición

Para cada $n \in \mathbb{N}$, el numeral de Church para n es un término $\underline{n}$ definido como

$$\underline{n} \equiv \lambda f\ x.\, f^n\ x$$

- Ejemplos

$$\underline{0} \equiv \lambda f\ x.\, x \qquad \underline{1} \equiv \lambda f\ x.\, f\ x \qquad \underline{2} \equiv \lambda f\ x.\, f\ (f\ x)$$

- Notar que $\underline{0} \equiv \mathit{False}$. No importa, ya que no usamos tipos.
- Definimos la función sucesor como

$$\mathit{succ} \equiv \lambda n.\lambda f\ x.n\ f\ (f\ x)$$

Funciones sobre numerales de Church

- ¿Cómo definir la suma?
- La suma $\underline{n+m}$ es aplicarle la función sucesor n veces a $\underline{m}$
- O sea que

$$suma \equiv \lambda n\ m.n\ succ\ m$$

Ejercicio

Definir una función isZero, tal que

$$\begin{aligned} isZero\ \underline{0} &=_\beta True \\ isZero\ \underline{n+1} &=_\beta False \end{aligned}$$

Funciones recursivas

- ¿Cómo definir una función recursiva?
- Por ejemplo queremos definir la función *fact*

$$fact \equiv_? \lambda n.\textbf{if } (isZero\ n) \textbf{ then } \underline{1} \textbf{ else } prod\ n\ (fact\ (pred\ n))$$

(suponemos ya definidas *prod* (producto) y *pred* (*predecesor*))

- Pero esto no es una definición válida! (¿Por qué?)
- Para resolver este problema se utilizan operadores de punto fijo, es decir operadores **F** tal que

$$\mathbf{F}\ X =_\beta X\ (\mathbf{F}\ X)$$

- Entonces

$$fact \equiv \mathbf{F}\ (\lambda f\ n.\textbf{if } (isZero\ n) \textbf{ then } \underline{1} \textbf{ else } prod\ n\ (f\ (pred\ n)))$$

Puntos fijos

- Considere el siguiente *combinador*:
  (un combinador es un término cerrado)

$$\mathbf{Y} \equiv \lambda x.(\lambda y.x\ (y\ y))\ (\lambda y.x\ (y\ y))$$

Teorema

Todo término X tiene un punto fijo dado por $(\mathbf{Y}\ X)$. Esto es:

$$\mathbf{Y}\ X =_\beta X\ (\mathbf{Y}\ X)$$

- Nota: $\mathbf{Y}$ no es el único operador de punto fijo (ver la práctica).

Resumen

- El λ-cálculo es un cálculo muy simple, pero muy poderoso.
- Ligadura de variables (binding)
- Nociones de reducción y de equivalencia.
- Estrategia de reducción normal
- Representación de booleanos, pares y naturales.
- Puntos fijos.
- ¡Es un lenguaje de programación!

Referencias

- *Lambda-Calculus and Combinators*. J. R. Hindley and J. P. Seldin. Cambridge University Press (2008).