

# Lenguajes y Compiladores

2016

## Estructura de la materia a grandes rasgos:

**Primera Parte:** Lenguaje imperativo

**Segunda Parte:** Lenguaje aplicativo puro, y lenguaje aplicativo con referencias y asignación

# Ejes de contenidos de la segunda parte

- 1 Cálculo Lambda

# El Cálculo Lambda

- Church (1930), continuado por Kleene, Rosser (1930s)  
**Motivación original:** problemas de fundamentación de la matemática
- **Como notación:** se usa para generar una expresión que denote una función, sin necesidad de dar nombre  
**Por ejemplo:**  $f(x) = x + y$  se puede escribir

$$f = \lambda x. x + y,$$

con lo cual uno se puede referir a la función  $f$  mediante la expresión  $\lambda x. x + y$ , sin necesidad de ponerle nombre.

## Sintaxis abstracta del CL

El CL puro sólo contiene variables, aplicaciones y la notación lambda (llamada abstracción).

$\langle exp \rangle ::=$	expresiones o términos
$\langle var \rangle$	variables
$  \langle exp \rangle \langle exp \rangle$	aplicación
$  \lambda \langle var \rangle . \langle exp \rangle$	abstracción o expresión lambda

**Convención:** La aplicación asocia a izquierda. Por ejemplo,

$\lambda x. (\lambda y. xy)x$  es lo mismo que

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

## Variables libres

En la abstracción

$$\lambda x.x + 2$$

se determina que estamos construyendo una función que varía de acuerdo a qué valor toma  $x$ . Es decir, la abstracción es un *ligador*.

$$FV(v) = \{v\}$$

$$FV(ee') = FV(e) \cup FV(e')$$

$$FV(\lambda v.e) = FV(e) - \{v\}$$

# Operador sustitución

**Conjunto de sustituciones:**  $\Delta = \langle var \rangle \rightarrow \langle exp \rangle$

**Operador sustitución:**  $_{-}/_{-} \in \langle exp \rangle \times \Delta \rightarrow \langle exp \rangle$

$$v/\delta = \delta v$$

$$(ee')/\delta = (e/\delta)(e'/\delta)$$

$$(\lambda v.e)/\delta = \lambda v_{new}. e/[\delta|v : v_{new}]$$

donde  $v_{new} \notin \bigcup_{w \in FV(e) - \{v\}} FV(\delta w)$

## Conversión $\alpha$

**Renombre:** Cambio en  $\lambda v.e$  de la variable ligada  $v$  (y todas sus ocurrencias) por una variable  $v'$  que no ocurra libre en  $e$ :

$$\lambda v'. e/v \mapsto v'$$

donde  $v' \notin FV(e)$ .

**$\alpha$ -conversión:** Si  $e_1$  se obtiene a partir de  $e_0$  por 0 o más renombres de ocurrencias de subfrases. También se dice que  $e_0$   $\alpha$ -convierte a  $e_1$ .

**Notación para expresiones  $\alpha$ -convertibles:**  $e_0 \equiv e_1$



## Ejecución: Contracción $\beta$

**Redex:** Es una expresión de la forma  $(\lambda v.e)e'$

**Contracción  $\beta$ :** Reemplaza en  $e_0$  una ocurrencia de un redex  $(\lambda v.e)e'$  por su contracción  $(e/v \mapsto e')$ , y luego efectúa cero o más renombres de cualquier subexpresión.

**Notación:** Si  $e_1$  es el resultado de una contracción  $\beta$  de  $e_0$ , entonces escribimos

$$e_0 \rightarrow e_1$$

## Ejecución: Formas normales

**Forma normal:** expresión sin redices.

Las formas normales representan configuraciones terminales.

Por eso la semántica operacional del cálculo lambda consiste en efectuar contracciones  $\beta$  hasta obtener formas normales.

**Ejemplo:**

$$(\lambda x. \lambda y. (\lambda x. xy)(zx))(yz) \rightarrow (\lambda x. \lambda y. (zx)y)(yz)$$

**Ejemplo:**

$$\begin{aligned} (\lambda x. \lambda y. (\lambda x. xy)(zx))(yz) &\rightarrow (\lambda x. \lambda y. (zx)y)(yz) \\ &\equiv (\lambda x. \lambda t. (zx)t)(yz) \end{aligned}$$

**Ejemplo:**

$$\begin{aligned} (\lambda x. \lambda y. (\lambda x. xy)(zx))(yz) &\rightarrow (\lambda x. \lambda y. (zx)y)(yz) \\ &\equiv (\lambda x. \lambda t. (zx)t)(yz) \\ &\rightarrow \lambda t. z(yz)t \end{aligned}$$

## Ejecución: hay expresiones divergentes

Sea

$$\Delta = \lambda x.xx$$

entonces  $\Delta\Delta$  no tiene forma normal:

$$(\lambda x.xx)(\lambda x.xx) \rightarrow (\lambda x.xx)(\lambda x.xx)$$

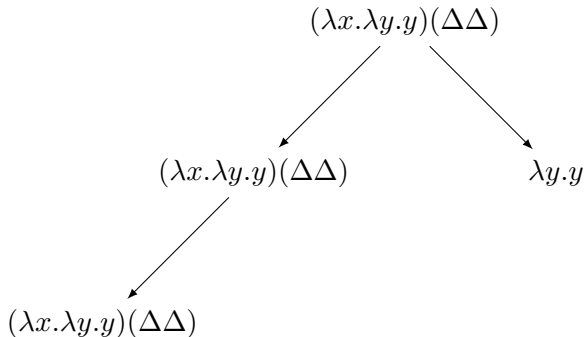
$$(\lambda x.xx)(\lambda x.xx) \rightarrow (\lambda x.xx)(\lambda x.xx)$$

$$\rightarrow (\lambda x.xx)(\lambda x.xx)$$

$$\vdots$$

## Ejecución: hay más de una forma de reducción

En otros casos, la forma normal se puede encontrar si elegimos correctamente qué redex reducir en cada caso:



Es decir la reducción no es determinística.

## Ejecución (formalmente)

$\rightarrow^*$  denota la clausura transitiva y reflexiva de  $\rightarrow$

(o sea, aplicar  $\rightarrow$  cero o más veces)

Por ejemplo, no existe  $n$  forma normal tal que  $\Delta\Delta \rightarrow^* n$

### Formalmente:

$e \rightarrow^* e'$  si y sólo si existen  $e_0, \dots, e_n$  (con  $n \geq 0$ ) tales que

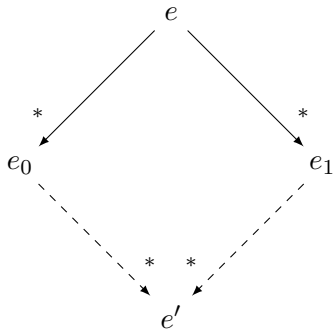
$$e = e_0 \rightarrow e_1 \rightarrow \dots \rightarrow e_n = e'$$

Notar que si  $n = 0$  entonces  $e = e'$

# Si existe forma normal, es única

Es consecuencia inmediata de:

**Teorema de Church-Rosser** Si  $e \rightarrow^* e_0$  y  $e \rightarrow^* e_1$ , entonces existe  $e'$  tal que  $e_0 \rightarrow^* e'$  y  $e_1 \rightarrow^* e'$ .



## Regla $\eta$

Un  $\eta$ -redex es una expresión de la forma  $\lambda v.ev$ , donde  $v \notin FV e$

$$\frac{}{\lambda v.ev \rightarrow e} \text{ si } v \notin FV e \quad (\eta)$$

# Evaluación

→\* no representa adecuadamente la ejecución de los lenguajes aplicativos.  
En los lenguajes aplicativos la evaluación

- 1 sólo para expresiones cerradas,
- 2 es determinística,
- 3 no busca formas normales sino formas canónicas.

Vamos a estudiar:

**Evaluación (en orden) normal:** lenguajes funcionales lazy (Haskell)

**Evaluación eager o estricta:** lenguajes estrictos (ML).



## Formas canónicas

La evaluación busca una forma canónica, las mismas juegan el rol de ser "valores" de expresiones.

La noción de forma canónica depende de la definición de evaluación. Se define una noción de forma canónica para la evaluación normal, y otra para la evaluación eager.

En el caso del cálculo lambda coinciden: **son las abstracciones**

## Formas canónicas vs. formas normales

**Propiedad:** Una aplicación cerrada no puede ser forma normal.

**Corolario:** una expresión cerrada que es forma normal es también forma canónica.

El recíproco no vale.

## Formas canónicas

¿Por qué conformarse con una forma canónica en vez de continuar ejecutando hasta obtener una forma normal?

Sólo tiene sentido evaluar expresiones cerradas.

Una vez que se alcanzó una abstracción  $\lambda v.e$ , continuar evaluando implicaría evaluar  $e$  que puede no ser una expresión cerrada, puede contener a la variable  $v$ .

## Semántica natural o big-step

En este tipo de semántica, uno no describe un paso de ejecución, sino directamente una relación entre los términos y sus valores (que también son términos, son formas canónicas).

Llamaremos  $\Rightarrow$  a esta relación.

# Evaluación Normal

## Reglas para $\Rightarrow_N$

Regla para las formas canónicas

$$\overline{\lambda v.e \Rightarrow_N \lambda v.e}$$

Regla para la aplicación

$$\frac{e \Rightarrow_N \lambda v.e_0 \quad (e_0/v \mapsto e') \Rightarrow_N z}{ee' \Rightarrow_N z}$$

# Evaluación Eager

## Reglas para $\Rightarrow_E$

Regla para las formas canónicas

$$\overline{\lambda v.e \Rightarrow_E \lambda v.e}$$

Regla para la aplicación

$$\frac{e \Rightarrow_E \lambda v.e_0 \quad e' \Rightarrow_E z' \quad (e_0/v \mapsto z') \Rightarrow_E z}{ee' \Rightarrow_E z}$$