



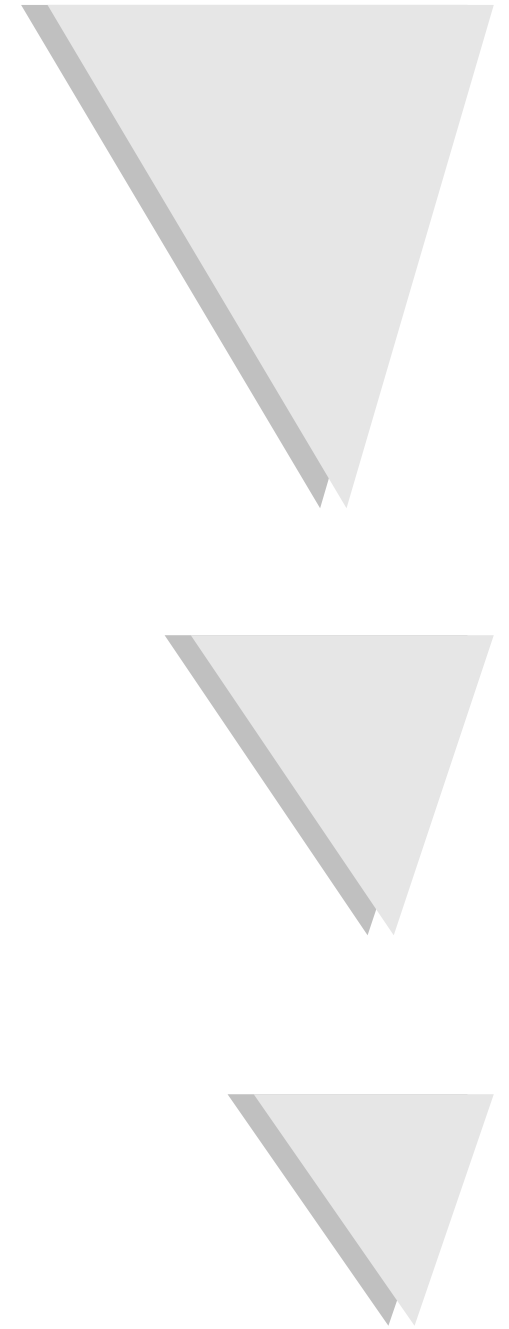
PROGRAMACIÓN FUNCIONAL

Lambda Cálculo: Semántica por Equivalencias



Lambda Cálculo

- ◆ Métodos para dar semántica
- ◆ Semántica por equivalencias
- ◆ Equivalencia entre λ -expresiones
 - ◆ α -equivalencia
 - ◆ β -reducción y β -equivalencia
 - ◆ η -equivalencia

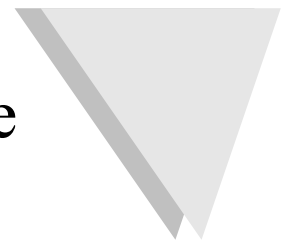
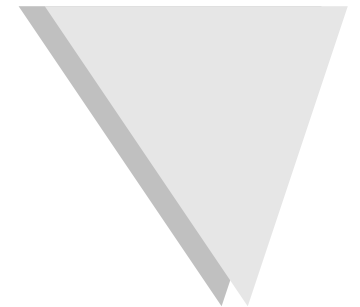
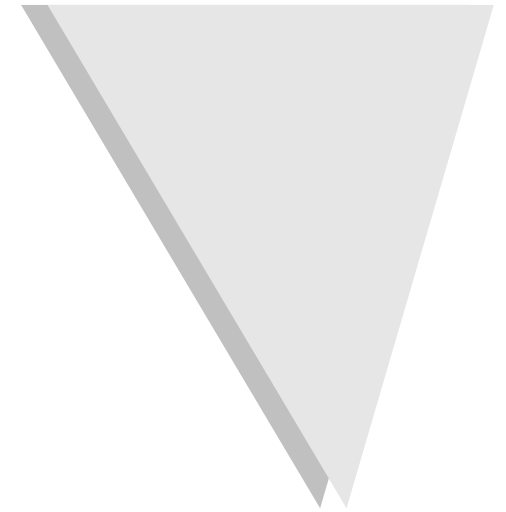


Lambda Cálculo

- ◆ ¿Cómo dar semántica a un lenguaje de programación?
 - ◆ Semántica algebraica
 - ◆ dar ecuaciones que indiquen cuando dos términos tienen el mismo significado (i.e. cocientar el conjunto de strings)
 - ◆ Semántica denotacional
 - ◆ dar una función que a cada término le asigna su significado (¡debe fijarse el conjunto de llegada!)
 - ◆ Semántica operacional
 - ◆ dar reglas que digan cómo calcular el resultado de computar un término (reducción o transición)

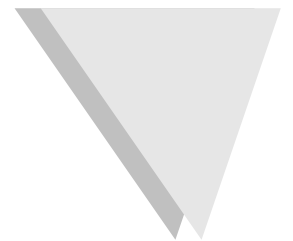
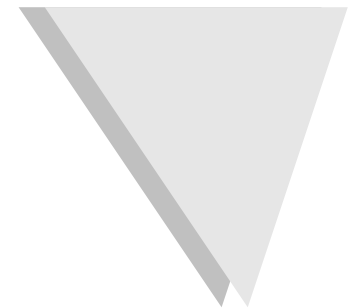
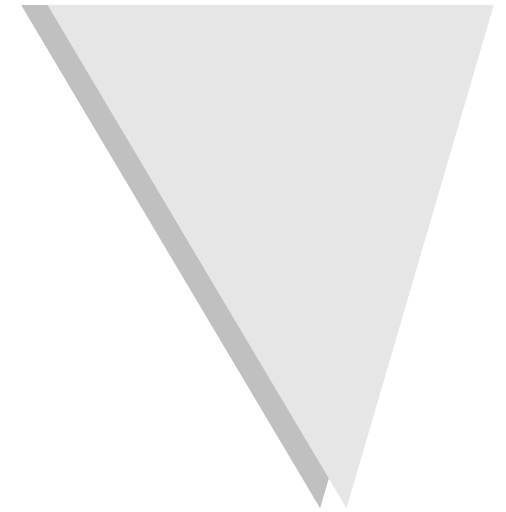
Lambda Cálculo

- ◆ ¿Cuál forma es más conveniente?
 - ◆ Depende del lenguaje
 - ◆ Puede usarse más de una
- ◆ ¿Y cómo sabemos que están bien?
 - ◆ Debe demostrarse que distintas formas de dar significado son equivalentes
 - ◆ puede hacerse en general, o en cada caso
- ◆ Estos temas en general corresponden a una materia de semántica de lenguajes de programación



Lambda Cálculo

- ◆ ¿Y para el λ -cálculo?
 - ◆ Veremos
 - ◆ una semántica algebraica
 - ◆ relaciones de equivalencia entre λ -expresiones
 - ◆ en particular, α , β y η -equivalencia
 - ◆ una semántica operacional
 - ◆ relaciones de reducción entre λ -expresiones
 - ◆ en particular β -reducción y η -reducción
- ◆ La semántica denotacional del λ -cálculo es muy compleja, y escapa al alcance del curso



Lambda Cálculo

◆ Renombre de variables

- ◆ El nombre de una variable ligada no es importante
- ◆ ¿Cómo capturar esto con una relación de equivalencia?
- ◆ Dos términos deben ser equivalentes si difieren sólo en el nombre de variables ligadas (intuitivamente representan a la misma función, y por ello queremos que sean equivalentes)
- ◆ Ejemplos:
 - ◆ $(\lambda x.x)$ es equivalente a $(\lambda y.y)$
 - ◆ $(\lambda xy.xy)$ es equivalente a $(\lambda zw.zw)$
 - ◆ pero $(\lambda x.xy)$ NO es equivalente a $(\lambda z.zw)$ (¿por qué?)

Lambda Cálculo

◆ Def: α -equivalencia (o α -conversión)

◆ Sea \approx_α la menor relación que satisface las siguientes reglas (M, N, P, x, y se asumen universalmente cuantificadas):

◆ si y no ocurre libre en M ,
entonces $(\lambda x.M) \approx_\alpha (\lambda y.M\{x \leftarrow y\})$ (axioma α)

◆ $M \approx_\alpha M$

◆ si $M \approx_\alpha N$ y $N \approx_\alpha P$, entonces $M \approx_\alpha P$

◆ si $M \approx_\alpha N$, entonces $N \approx_\alpha M$

◆ si $M \approx_\alpha N$,

entonces $MP \approx_\alpha NP$, $PM \approx_\alpha PN$ y $\lambda x.M \approx_\alpha \lambda x.N$

Lambda Cálculo

- ◆ Consideremos el conjunto Λ / \approx_α , definido por

$$\Lambda / \approx_\alpha = \{ \{ N / N \in \Lambda \wedge N \approx_\alpha M \} \mid M \in \Lambda \}$$

- ◆ Los elementos son conjuntos de λ -términos α -equivalentes (llamados *α -clases de equivalencia*)
 - ◆ Ej: $\{ \lambda x.x, \lambda y.y, \lambda z.z, \lambda w.w, \dots \}$
- ◆ Todos los elementos de un conjunto significan lo mismo
- ◆ Por lo tanto, podemos usar cualquiera de ellos indistintamente (elegimos el *representante* que más nos conviene)
- ◆ Podemos denotar un conjunto mediante uno cualquiera de sus representantes

Lambda Cálculo

- ◆ De aquí en más, trabajaremos con elementos de Λ/\approx_α como si fueran elementos de Λ

- ◆ O sea, permitiremos renombrar las variables ligadas siempre que sea conveniente
- ◆ Simplifica las definiciones posteriores

- ◆ **Hipótesis de Barendregt:**

todas las variables ligadas de un λ -término son distintas entre sí, y distintas de todas sus variables libres

- ◆ ¡Esta hipótesis puede hacerse pues siempre existe un representante que cumple la condición!

Lambda Cálculo

◆ Aplicación funcional

- ◆ La expresión que aplica una función a su argumento denota el mismo valor que el resultado
- ◆ ¿Cómo expresamos esta propiedad con una equivalencia?
- ◆ Dos términos deben ser equivalentes si se pueden cambiar aplicaciones por sus resultados
- ◆ Ejemplos:
 - ◆ $(\lambda f x. f x)(\lambda z. z)y$ es equivalente a $(\lambda x. (\lambda z. z)x)y$
 - ◆ $(\lambda f x. f x)(\lambda z. z)y$ es equivalente a $(\lambda z. z)y$
 - ◆ $(\lambda w. w)y$ es equivalente a y

Lambda Cálculo

◆ Def: β -equivalencia (o β -conversión)

- ◆ Sea \approx_β la menor relación que satisface las siguientes reglas
(M, N, P, x se asumen universalmente cuantificadas):

- ◆ $(\lambda x.M)N \approx_\beta M\{x \leftarrow N\}$ (axioma β)

- ◆ $M \approx_\beta M$

- ◆ si $M \approx_\beta N$ y $N \approx_\beta P$, entonces $M \approx_\beta P$

- ◆ si $M \approx_\beta N$, entonces $N \approx_\beta M$

- ◆ si $M \approx_\beta N$,
entonces $MP \approx_\beta NP$, $PM \approx_\beta PN$ y $\lambda x.M \approx_\beta \lambda x.N$

Lambda Cálculo

Observaciones

- Los términos M , N , etc. utilizados en la definición son α -clases de equivalencias
(o equivalentemente, se puede usar la hipótesis de Barendregt)
- La primer regla hace que una aplicación de función y su resultado sean equivalentes
- La sustitución se utiliza para modelar el cambio de un parámetro formal por uno real

Ejemplos:

- $g(gz) \approx_{\beta} (\lambda f. \lambda x. f(fx))(\lambda y. gy)z \approx_{\beta} (\lambda h. h(hz))g$
- $(\lambda x. xx)(\lambda z. z) \approx_{\beta} (\lambda z. z)(\lambda w. w) \approx_{\beta} (\lambda y. y)$

Lambda Cálculo

- ◆ Consideremos el conjunto $\Lambda / \approx_\alpha \approx_\beta$, definido por

$$\Lambda / \approx_\alpha \approx_\beta = \{ \cup \{ \mathcal{P} \mid \mathcal{P} \in \Lambda / \approx_\alpha \wedge \mathcal{P} \approx_\beta \mathcal{Q} \} \mid \mathcal{Q} \in \Lambda / \approx_\alpha \}$$

- ◆ Los elementos son conjuntos de λ -términos $\alpha\beta$ -equivalentes (llamados $\alpha\beta$ -*clases de equivalencia*)
 - ◆ Ej: $\{ \lambda x.x, (\lambda y.y)(\lambda z.z), (\lambda wx.wx)(\lambda z.z)(\lambda u.u), \dots \}$
- ◆ Todos los elementos de un conjunto significan lo mismo
- ◆ Podemos elegir el *representante* que más nos conviene
- ◆ Pero, no todos los elementos son iguales en su forma...
¿cuál nos convendrá utilizar?

Lambda Cálculo

- ◆ De manera equivalente
 - ◆ definir una relación $\approx_{\alpha\beta}$ y el conjunto $\Lambda/\approx_{\alpha\beta}$
 - ◆ luego mostrar que $\Lambda/\approx_{\alpha\beta} = \Lambda/\approx_{\alpha} \approx_{\beta}$
- ◆ ¿Alcanza $\Lambda/\approx_{\alpha\beta}$ para dar significado a Λ ?
 - ◆ O sea, ¿cada elemento de $\Lambda/\approx_{\alpha\beta}$, representa a una función diferente?
 - ◆ No. Por ejemplo $(\lambda x.f x)$ no es $\alpha\beta$ -equivalente a f
 - ◆ ¡sin embargo, para todo M , $((\lambda x.f x)M) \approx_{\alpha\beta} (fM)$!

Lambda Cálculo

◆ Extensionalidad

- ◆ Queremos que dos funciones sean iguales si y sólo si dan el mismo resultado al ser aplicadas al mismo valor
- ◆ ¿Cómo expresamos esta propiedad con una equivalencia?
- ◆ Dos términos deben ser equivalentes si pueden cambiar un término M por uno de la forma $(\lambda x.Mx)$ (siempre que x no ocurra libre en M)
- ◆ Ejemplos:
 - ◆ $(\lambda x.fx)$ es equivalente a f
 - ◆ $(\lambda z.(\lambda y.y)z)$ es equivalente a $(\lambda y.y)$
 - ◆ $(\lambda w.(\lambda y.wy))$ es equivalente a $(\lambda w.w)$

Lambda Cálculo

◆ Def: η -equivalencia (o η -conversión)

◆ Sea \approx_η la menor relación que satisface las siguientes reglas (M, N, P, x se asumen universalmente cuantificadas):

◆ si x no ocurre libre en M ,
entonces $(\lambda x.Mx) \approx_\eta M$ (axioma η)

◆ $M \approx_\eta M$

◆ si $M \approx_\eta N$ y $N \approx_\eta P$, entonces $M \approx_\eta P$

◆ si $M \approx_\eta N$, entonces $N \approx_\eta M$

◆ si $M \approx_\eta N$,
entonces $MP \approx_\eta NP$, $PM \approx_\eta PN$ y $\lambda x.M \approx_\eta \lambda x.N$

Lambda Cálculo

- ◆ En realidad
 - ◆ definir una relación $\approx_{\alpha\beta\eta}$ y el conjunto $\Lambda/\approx_{\alpha\beta\eta}$
- ◆ ¿Alcanza $\Lambda/\approx_{\alpha\beta\eta}$ para dar significado a Λ ?
 - ◆ ¿Cada elemento de $\Lambda/\approx_{\alpha\beta\eta}$ representa algo diferente?
 - ◆ No. Sólo funciona para los términos definidos (o sea, cuya computación termina)
 - ◆ Para los otros, hay que hacer algunas definiciones extras, y juntar todos los que no terminan
 - ◆ Depende de la estrategia de reducción (¡semántica estricta o no estricta!)

Resumen

- ◆ Las semántica de lenguajes de programación es un tema muy importante
- ◆ Existen diversos métodos para dar semántica
- ◆ El más usado en λ -cálculo es cocientar las expresiones mediante *equivalencias* adecuadas
- ◆ Estas equivalencias son 3:
 - ◆ α -equivalencia: modela el renombre de variables
 - ◆ β -equivalencia: modela la reducción
 - ◆ η -equivalencia: modela la extensionalidad