.UBAfiuba
FACULTAD DE INGENIERÍA

Cálculo Lambda

TB025 - Paradigmas de Programación

Curso Essaya

Origen

En la década de 1930, varios matemáticos estaban interesados en la pregunta: ¿cuándo una función $f : \mathbb{N} \to \mathbb{N}$ es **computable**?

Es decir, ¿es posible calcular $f(n)$ para cualquier n dado, usando lápiz y papel?

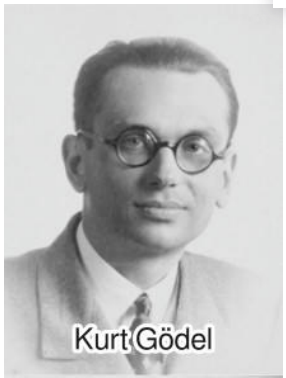
Se desarrollaron tres modelos de computabilidad por separado:

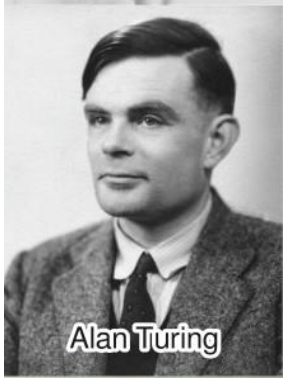
Kurt Gödel: Definió las **Funciones Recursivas Generales** y postuló que una función es computable si y solo si es recursiva general.

Alan Turing: Definió la **Máquina de Turing** y postuló que una función es computable si y solo si es computable por una máquina de Turing.

Alonzo Church: Definió el **Cálculo Lambda** y postuló que una función es computable si y solo si puede escribirse como una expresión lambda.

Luego se demostró que estos tres modelos son equivalentes (**Tesis de Church-Turing**).





Funciones

El Cálculo Lambda incorpora dos simplificaciones para operar con funciones:

1. **Las funciones son anónimas.** Por ejemplo, la función

$$\text{suma\_cuadrados}(x, y) = x^2 + y^2$$

   se puede escribir como

$$(x, y) \mapsto x^2 + y^2$$

2. **Todas las funciones son de un solo argumento.** Por ejemplo, la función

$$(x, y) \mapsto x^2 + y^2$$

   se puede escribir como

$$x \mapsto (y \mapsto x^2 + y^2)$$

   Esta transformación se llama *currificación* (por Haskell Curry).

Cálculo Lambda

Consiste en escribir **expresiones lambda** con una sintaxis formal, y aplicar **reglas de reducción** para transformarlas.

Una **expresión lambda** puede ser:

- Una **variable**: `x`, `y`, `z`, ...
- Una **abstracción**: `(λx.M)` donde `M` es una expresión lambda.
- Una **aplicación**: `(M N)` donde `M` y `N` son expresiones lambda.

Ejemplo: `(((λx.(x y)) (λy.(y y))) z)`

Una **abstracción** `(λx.M)` denota una **función anónima** que toma un argumento `x` y devuelve `M`, es decir $x \mapsto M$.

Una **aplicación** `(M N)` denota la acción de **invocar** la función `M` con el argumento `N`, es decir $M(N)$.

Convenciones

Para simplificar la notación:

1. Se pueden omitir los paréntesis externos: `M N ≡ (M N)`
2. Las aplicaciones se asocian a la izquierda: `M N P ≡ ((M N) P)`
4. El cuerpo de la abstracción se extiende todo lo posible hacia la derecha: `λx.M N ≡ λx.(M N)`
5. Se pueden contraer múltiples abstracciones lambda: `λx.λy.λz.N ≡ λx y z.N`
3. Cuando todas las variables son de una única letra, se pueden omitir los espacios: `MNP ≡ M N P`

Ejemplo: para simplificar `(((λx.(λy.(y x))) a) b)`:

- `((λx.(λy.(y x))) a) b` (1)
- `(λx.(λy.(y x))) a b` (2)
- `(λx.λy.y x) a b` (3)
- `(λx y.y x) a b` (4)
- `(λxy.yx)ab` (5)

Variables libres y ligadas

Una abstracción `λx.M` **liga** las variables `x` que aparecen en el cuerpo `M`. Todas las demás variables en `M` son **libres**.

```
(λz.z x y) (λx.x)
```

Reglas de reducción

Conversión Alfa (α): cambiar variables ligadas

> En la abstracción `λx.M`, se puede cambiar `x` por cualquier otra variable que no aparezca libre en `M`.
>
> Por ejemplo, `λx.x y ≡ λz.z y`.

Reducción Beta (β): aplicar una β-redex[1]

> Una **β-redex** es una aplicación de la forma `(λx.M) N`.
>
> Por ejemplo, `(λu.u z u) a ≡ a z a`.

Reducción Eta (η): Reducir una η-redex

> Una **η-redex** es una expresión de la forma `λx.M x` donde `x` no aparece libre en `M`.
>
> Según la regla η, `λx.M y ≡ M`.

[1] "redex" = expresión reducible.

www.**ingenieria**.uba.ar

/ingenieriauba

/FIUBAoficial