Teórica 4 : Cálculo λ

Tomás Felipe Melli

April 27, 2025

Índice

1 Introducción

1.1 Qué es Cálculo λ ?

El **Cálculo** λ es un sistema creado en los años 30 por Alonzo Church (matemático y lógico estadounidense) para describir cómo funcionan las funciones de manera completamente formal y matemática. Church quería entender qué significa computar algo, es decir, qué puede hacer una máquina de manera mecánica. Como consecuencia, creó este sistema. Que más adelante, en los 60, se comenzo utilizar para estudiar semántica formal de lenguajes de programación. En la actualidad es la base de muchos lenguajes funcionales. La importancia de este sistema yace en que muestra que todo programa se puede pensar simplemente como funciones que toman datos y devuelven resultados.

2 Cálculo λ^b : sintaxis y tipado

2.1 Cálculo λ^b

2.1.1 Sintaxis de los tipos

En este caso, vamos a ver cómo se comportan los **booleanos**. Por ello definimos

$$\tau, \sigma, \rho, ... ::= bool$$
$$\mid \quad \tau \to \sigma$$

O sea, que puede ser simplemente un booleano o una función que va de un tipo τ a uno σ. Esta función que la escribimos con el operador "$\to$" es **asociativo a derecha** de forma que :

$$\tau \to \sigma \to \rho = \tau \to (\sigma \to \rho) \neq (\tau \to \sigma) \to \rho$$

Es decir que, toma un argumento de tipo τ y devuelve una función que toma un argumento de tipo σ y devuelve algo de tipo ρ.
Es importante destacar que tanto τ como σ si bien sólo tenemos definido el tipo *bool*, si consideramos $\tau \to \sigma$ puede ser algo del tipo $(bool \to bool) \to bool$ que también tenemos definido en nuestro sistema. Como otras tantas formas.

2.1.2 Sintaxis de los Términos

Supongamos dado un conjunto **inifinito numerable de variables**

$$\chi = \{x, y, z, ...\}$$

** Nos detenemos por un momento para detallar por qué es importante tener un conjunto de estas características para lo que sigue.

- Que sea **inifinito** es importante ya que tendremos infinitas variables para construir funciones o expresiones, ya que si tuviesemos finita cantidad de ellas sería un problema introducir nuevas. Por otro lado, al asignar un nuevo valor podríamos romper el significado de otras expresiones (renombrar adecuadamente y evitar colisiones).
- Que sea **numerable** es también importante ya que nos garantiza que se puedan **listar uno tras otro** y como consecuencia, utilizar **inducción**.

Ahora sí, pasemos a ver cómo se construyen los términos del sistema.

$M, N, P, \ldots$	::=	x	variable
		$\mid \lambda x : \tau . M$	abstracción
		$\mid M\ N$	aplicación
		$\mid$ true	verdadero
		$\mid$ false	falso
		$\mid$ if M then N else P	condicional

Para poder construir términos en cálculo λ tenemos las siguientes reglas.

1. Podemos tener un término que sea una **variable** como x.
2. Podemos tener un término que sea una **abstracción** como $\lambda x : \tau . M$ que representa la función que toma un argumento x que es del tipo τ y el cuerpo de la función es M.

3. Podemos tener un término que sea la **aplicación** de un término a otro. Que significa básicamente que M será una función que se le aplicará a N que será el argumento que se le pase a dicha función. Estamos representando *aplicar la función.*

4. Podemos tener términos **constantes** como **true** y **false** que representan los valores booleanos verdadero y falso.

5. Podemos tener un término **condicional** donde M es una expresión booleana y según su valor de verdad evaluará a N o a P.

En el caso de la **aplicación**, asociamos a **izquierda**.

$$M\ N\ P \quad = (M\ N)\ P \neq M\ (N\ P)$$

Es decir que siempre que tengamos una aplicación a varios términos, vamos a agrupar de **izquierda a derecha**. En este caso, M aplica a N y luego ese resultado se aplica a P.
La **abstracción** y el **condicional** tienen **menor precedencia** que la **aplicación**:

$$\lambda x : \tau.M\ N = \lambda x : \tau.(M\ N) \neq (\lambda x : \tau.M)\ N$$

La **precedencia** es el **orden en que se aplican las operaciones**. En este escenario, primero ocurre la aplicación de M a N, y a continuación, la abstracción. La verdad que este ejemplo podría confundir, miremos el siguiente:

$$(\lambda x : \tau.\ If\ x\ then\ true\ else\ false)\ M$$

En este escenario, la de **mayor precedencia** es la **aplicación**. Por tanto, sustituimos a x por M (aplicación de la abstracción):

$$\lambda x : \tau.\ If\ M\ then\ true\ else\ false$$

A continuación evaluamos el condicional:

$$If\ M\ then\ true\ else\ false$$

2.2 Variables libres y ligadas

Una **ocurrencia** de x está **ligada** si aparece dentro de una *abstracción* λx. Una ocurrencia de x está **libre** si no está ligada. Ejemplo :

$$(\lambda x : bool \to bool.\lambda y : bool.x\ y)\ (\lambda y : bool.x\ y)\ y$$

Un detalle importante es el **scope**. Ya que en este escenario el primer paréntesis tiene ambas x *e* y ligadas a abstracciones. Cosa que en el segundo no pasa ya que x está libre. Finalmente, y no está ligada a ninguna abstracción.
El **conjunto de variables libres** $fv(M)$ (por free-variables) de M es el conjunto de todas las variables que aparecen en la expresión M que no están ligadas a ninguna abstracción dentro de M,

$$\begin{aligned} fv(x) &\stackrel{def}{=} \{x\} \\ fv(\text{true}) = fv(\text{false}) &\stackrel{def}{=} \emptyset \\ fv(\text{if } M \text{ then } N \text{ else } P) &\stackrel{def}{=} fv(M) \cup fv(N) \cup fv(P) \\ fv(M\ N) &\stackrel{def}{=} fv(M) \cup fv(N) \\ fv(\lambda x : \tau.M) &\stackrel{def}{=} fv(M) \setminus \{x\} \end{aligned}$$

2.3 α-equivalencia

Decimos que dos términos M y N que difieren solamente en el nombre de sus variables ligadas son $\alpha - equivalentes$ (tiene una relación $=_\alpha$). Esto es una **relación de equivalencia**.

$$\lambda x : \tau.\lambda y : \sigma.x\ =_\alpha \lambda y : \tau.\lambda x : \sigma.y =_\alpha \lambda a : \tau.\lambda b : \sigma.a$$

Estas expresiones son equivalentes ya que todas se comportan de la misma manera, toman dos argumentos, uno de tipo τ y otro de tipo σ y retorna x. El nombre de las variables ligadas es lo que cambia.
Veamos un ejemplo diferente :

$$\lambda x : \tau.\lambda y : \sigma.x\ \neq_\alpha \lambda x : \tau.\lambda y : \sigma.y =_\alpha \lambda x : \tau.\lambda x : \sigma.x$$

Para la primer expresión tenemos algo como lo que vimos antes, su comportamiento es tomar dos argumentos y devuelve x. Sin embargo, en segunda expresión sucede que no devuelve a x, sino que devuelve y por lo que se comporta diferente. Con respecto a la tercera, esta es $\alpha - equivalente$ a la segunda y esto ocurre ya que el comportamiento de devolver y es sólo un renombre.

2.4 Sistema de Tipos

El sistema de tipos es la parte formal que establece qué tipos pueden tener las variables y las expresiones. El concepto de **tipabilidad** se refiere a la capacidad de asignar un tipo a una expresión. Esta asignación se formaliza mediante un sistema deductivo, que proporciona reglas para deducir qué tipo tiene una expresión en función de su estructura.

2.4.1 Contextos de tipado

Un **contexto de tipado** es un conjunto finito de pares $(x_i : \tau_i)$

$$\{x_1 : \tau_1, ..., x_n : \tau_n\}$$

donde :

- $x_1, x_2, ..., x_n$ son variables
- $\tau_1, \tau_2, ..., \tau_n$ son los tipos asociados a las anteriores

Un contexto de tipado tiene la restricción de que no puede haber variables repetidas ($i \neq j \Rightarrow x_i \neq x_j$). Lo notamos con letras griegas mayúsculas $(\Gamma, \Delta, ...)$ y a veces notamos $dom(\Gamma) = \{x_1, ..., x_n\}$ como el **dominio del contexto** que simplemente es el conjunto de variables que tiene Γ y qué tipo les asigna.

2.4.2 Juicios de tipado

El sistema de tipos hace afirmaciones sobre **juicios de tipado** de la forma

$$\Gamma \vdash M : \tau$$

2.4.3 Reglas de tipado

$$\frac{}{\Gamma \vdash \mathit{true} : \mathit{bool}} \text{ T-TRUE}$$

$$\frac{}{\Gamma \vdash \mathit{false} : \mathit{bool}} \text{ T-FALSE}$$

$$\frac{\Gamma \vdash M : \mathit{bool} \quad \Gamma \vdash N : \tau \quad \Gamma \vdash P : \tau}{\Gamma \vdash \mathit{if}\ M\ \mathit{then}\ N\ \mathit{else}\ P : \tau} \text{ T-IF}$$

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \text{ T-VAR}$$

$$\frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x : \tau . M : \tau \to \sigma} \text{ T-ABS}$$

$$\frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash M\ N : \sigma} \text{ T-APP}$$

2.4.4 Propiedades del sistema de tipos

- **Unicidad de tipos** (Teorema)

  Si $\Gamma \vdash M : \tau$ *y* $\Gamma \vdash M : \sigma$ son derivables, entonces $\tau = \sigma$

- **Weakening + Strenghtening**

  Si $\Gamma \vdash M : \tau$ es derivable y $fv(M) \subseteq dom(\Gamma \cap \Gamma')$ entonces $\Gamma' \vdash M : \tau$ es derivable.

3 Cálculo λ^b : semántica operacional

El sistema de tipos indica cómo se construyen los programas. Queremos ahora darles **significado** (semántica). Hay diferentes maneras:

1. **Semántica operacional**
   La semántica operacional describe cómo se ejecutan los programas, es decir, cómo los términos del programa se evalúan hasta llegar a un resultado. Existen dos enfoques principales en la semántica operacional:

   (a) **Semántica Small-Step** : En la semántica small-step, la ejecución del programa ocurre paso a paso, evaluando el programa de una forma que refleja los cambios a lo largo de la ejecución, pero no necesariamente hasta el resultado final en un solo paso.

   (b) **Semántica Big-Step** : La semántica big-step describe cómo se evalúa directamente un programa hasta su resultado final. No se detalla el proceso paso a paso, sino que se especifica la evaluación completa.

2. **Semántica denotacional**
   La semántica denotacional interpreta los programas como objetos matemáticos que tienen una representación matemática. En lugar de describir cómo se ejecutan los programas, la semántica denotacional describe lo que un programa "significa" en términos matemáticos. La semántica denotacional no se enfoca en los detalles de ejecución, sino en la interpretación matemática del programa.

3. **Semántica axiomática**
   La semántica axiomática establece relaciones lógicas entre el estado del programa antes y después de la ejecución. El objetivo es especificar qué propiedades lógicas deben cumplirse al ejecutar el programa. Esto se hace mediante reglas que describen cómo los estados del programa cambian con cada operación. Esta semántica es útil para la verificación formal de programas, ya que permite razonar sobre el comportamiento de un programa en términos de su estado lógico.

4. **...**

3.1 Small-Step

Vamos a trabajar con la semántica operacional small-step.

3.1.1 Programas

Un **programa** es un término M **tipable** y cerrado. Esto quiere decir que cumple que

1. Dado un contexto de tipos Γ, existe un tipo τ tal que $\Gamma \vdash M : \tau$. Es decir que M puede ser derivado o probado dentro de un sistema de tipado, y tiene un tipo válido.

2. Todas las variables en M están ligadas y por ello notamos $fv(M) = \emptyset$

Decimos que el juicio $\Gamma \vdash M : \tau$ debe ser derivable para algún τ.

3.1.2 Juicios de evaluación

La semántica operacional hace afirmaciones sobre **juicios de evaluación**

$$M \to N$$

donde M y N son programas. Significa que el término M se puede evaluar en un término N . Es decir, al ejecutar el término M , llegamos a un nuevo término N que es el resultado de esa evaluación.

3.1.3 Valores

Los **valores** son los posibles resultados de evaluar programas:

$$V ::= true \mid false \mid \lambda x : \tau.M$$

3.1.4 Reglas de evaluación para expresiones booleanas

$$\frac{}{\text{if } \mathbf{true} \text{ then } M \text{ else } N \to M}\ \text{E-IFTRUE}$$

$$\frac{}{\text{if } \mathbf{false} \text{ then } M \text{ else } N \to N}\ \text{E-IFFALSE}$$

$$\frac{M \to M'}{\text{if } M \text{ then } N \text{ else } P \to \text{ if } M' \text{ then } N \text{ else } P}\ \text{E-IF}$$

Lo que nos dice $\to$ es que podemos reducir la expresión (evaluar) a la expresión que indica.
Ejemplos :

1. Derivar :
   if (if **false** then **false** else **true**) then **false** else **true**.
   Si evaluamos primero el if interno, que según **E-IFFALSE** evaluamos a **true**, nos quedaría if **true** then **false** else **true** que luego por **E-IFTRUE** evaluamos a **false**.

2. Para qué términos M vale que $true \to M$? Para cualquiera, **E-IFTRUE**

3. Es posible evaluar
   if true then (if false then false else false) else true.
   Por **E-IFFALSE** nos queda el if interno evaluado a **false** y si aplicamos **E-IFTRUE** evaluamos a **false**.

3.1.5 Reglas de evaluación para funciones (abstracción y aplicación)

$$\frac{M \to M'}{M\ N \to M'\ N}\ \text{E-APP1}$$

$$\frac{N \to N'}{(\lambda x : \tau.M)\ N \to (\lambda x : \tau.M)\ N'}\ \text{E-APP2}$$

$$\frac{}{(\lambda x : \tau.M)\ V \to M\{x := V\}}\ \text{E-APPABS}$$

3.2 Sustitución

La operación de **sustitución**

$M\{x := N\}$

denota el término que resulta de reemplazar todas las ocurrencias libres de x en M por N

3.2.1 Definición

$$x\{x := N\} \stackrel{def}{=} N$$

$$a\{x := N\} \stackrel{def}{=} a \text{ si } a \in \{true, false\} \cup \chi \setminus \{x\}$$

$$(\text{if } M \text{ then } P \text{ else } Q)\{x := N\} \stackrel{def}{=} \text{if } M\{x := N\} \text{ then } P\{x := N\} \text{ else } Q\{x := N\}$$

$$(M_1 M_2)\{x := N\} \stackrel{def}{=} M_1\{x := N\}M_2\{x := N\}$$

$$(\lambda y : \tau.M)\{x := N\} \stackrel{def}{=} \begin{cases} \lambda y : \tau.M & \text{si } x = y \\ \lambda y : \tau.M\{x := N\} & \text{si } x \neq y, y \notin \text{fv}(N) \\ \lambda z : \tau.M\{y := z\}\{x := N\} & \text{si } x \neq y, y \in \text{fv}(N), \\ & z \notin \{x, y\} \cup \text{fv}(M) \cup \text{fv}(N) \end{cases}$$

3.2.2 Definición alternativa

$$x\{x := N\} \stackrel{def}{=} N$$

$$a\{x := N\} \stackrel{def}{=} a \; si \; a \in \{true, false\} \cup \chi \setminus \{x\}$$

$$(\text{if } M \text{ then } P \text{ else } Q)\{x := N\} \stackrel{def}{=} \text{if } M\{x := N\} \text{ then } P\{x := N\} \text{ else } Q\{x := N\}$$

$$(M_1 M_2)\{x := N\} \stackrel{def}{=} M_1\{x := N\}M_2\{x := N\}$$

$$(\lambda y : \tau.M)\{x := N\} \stackrel{def}{=} \lambda y : \tau.M\{x := N\} \text{ asumiendo y } \notin \{x\} \cup fv(N)$$

La suposición se puede cumplor siempre, renombrando la variable ligada "y" en caso de conflicto.
Hagamos el siguiente ejercicio de ejemplo de una evaluación

$$(\lambda x : bool.\lambda f : bool \to bool.f \; (f \; x)) \; true \; (\lambda x : bool.x)$$

1. Aplicamos la función a **true**

   $$= (\lambda f : bool \to bool.f \; (f \; true)) \; (\lambda x : bool.x)$$

2. Aplicamos la función al argumento $(\lambda x : bool.x)$

   $$= (\lambda x : bool.x) \; ((\lambda x : bool.x) \; true))$$

3. Aplicamos la función del segundo término a **true**

   $$= (\lambda x : bool.x) \; (true)$$

4. Aplicamos finalmente a **true**

   $$= true$$

3.3 Propiedades de la evaluación

- **Determinismo** (Teorema)

  $Si \; M \to N_1 \; y \; M \to N_2 \; entonces \; N_1 = N_2$

- **Preservación de tipos** (Teorema)

  $Si \vdash M : \tau \; y \; M \to N \; entonces \vdash N : \tau$

- **Progreso**(Teorema)

  $Si \vdash M : \tau \; entonces$:

  1. O bien M es un **valor**
  2. O bien existe N tal que $M \to N$

- **Terminación** (Teorema)

  $Si \; \vdash M : \tau, \; entonces \; no \; hay \; una \; cadena \; infinita \; de \; pasos$:

  $$M \to M_1 \to M_2 \to ...$$

- **Corolario** (Canonicidad)

  1. Si $\vdash M : bool$ es derivable, entonces la evaluación de M termina y el resultado es $true$ o $false$
  2. Si $\vdash M : \tau \to \sigma$ es derivable, entonces la evaluación de M termina y el resultado es una **abstracción**

- **Slogan** *Well typed programs cannot go wrong* (Robin Milner)

- **Forma normal** Una **forma normal** es un término que no puede evaluarse más. (M tal que no existe $N, M \to N$)

- **Lema Todo valor está en su forma normal**

- **Estado de Error** Estado de evaluación, donde el término **está en forma normal pero** no es un valor. Representa estado en el cual el sistema de *runtime* en una implementación real generaría una excepción. Recordar que un valor es el resultado al que puede evaluar un término bien-tipado y cerrado

3.4 Evaluación en muchos pasos

3.4.1 Juicio en muchos pasos

La evaluación en muchos pasos $\twoheadrightarrow$ (también denotado $\rightarrow^*$) es la clausura reflexiva-transitiva de $\rightarrow$.

1. Si $M \rightarrow M'$, entonces $M \twoheadrightarrow M'$
2. $M \twoheadrightarrow M$ para todo M
3. Si $M \twoheadrightarrow M'$ y $M' \twoheadrightarrow M''$, entonces $M \twoheadrightarrow M''$

   if **true** then (if **false** then **false** else **true**) else **true** $\twoheadrightarrow$ **true**

3.4.2 Propiedades

Para el cálculo de expresiones booleanas valen:

- **Unicidad de formas normales**(Lema)

  Si $M \twoheadrightarrow U$ y $M \twoheadrightarrow V$ con V formas normales, entonces $U = V$

- **Terminación**(Lema)

  Par todo M existe una forma normal N tal que $M \twoheadrightarrow N$

4 Cálculo λ^{bn} : extensión con números naturales

4.1 Sintaxis

4.1.1 Tipos

$\tau, \sigma, ... \ ::= \ ... \mid nat$

4.1.2 Términos

Semántica informal

M	$::=$	$\ldots$
	$\mid zero$	el número cero
	$\mid succ(M)$	el sucesor del número que representa M
	$\mid pred(M)$	el predecesor del número que representa M
	$\mid isZero(M)$	representa un booleano **true/false**, dependiendo si M representa al cero

4.2 Reglas de tipado

$$\frac{}{\Gamma \vdash zero : nat}\ \text{T-ZERO}$$

$$\frac{\Gamma \vdash M : nat}{\Gamma \vdash succ(M) : nat}\ \text{T-SUCC}$$

$$\frac{\Gamma \vdash M : nat}{\Gamma \vdash pred(M) : nat}\ \text{T-PRED}$$

$$\frac{\Gamma \vdash M : nat}{\Gamma \vdash isZero(M) : bool}\ \text{T-ISZERO}$$

4.3 Valores

Extendemos el conjunto de **valores :**

$V ::= \ldots \mid zero \mid succ(V)$

4.4 Semántica operacional

$$\frac{M \to M'}{succ(M) \to succ(M')} \text{ E-SUCC}$$

$$\frac{M \to M'}{pred(M) \to pred(M')} \text{ E-PRED}$$

$$\frac{}{pred(succ(V)) \to V} \text{ E-PREDSUCC}$$

$$\frac{M \to M'}{isZero(M) \to isZero(M')} \text{ E-ISZERO}$$

$$\frac{}{isZero(zero) \to true} \text{ E-ISZEROZERO}$$

$$\frac{}{isZero(succ(V)) \to false} \text{ E-ISZEROSUCC}$$

Ejemplo:

1. $isZero(succ(pred(succ(zero)))$

   $\overset{E-PREDSUCC}{=} isZero(succ(zero))$

   $\overset{E-ISZEROSUCC}{=} false$

2. $isZero(pred(pred(succ(zero))))$

   $\overset{E-PREDSUCC}{=} isZero(pred(zero))$

   $\overset{ERROR}{=} \bot$

4.4.1 Forma normal ("f.n.")

Un programa M es una **f.n.** si no existe M' tal que $M \to M'$.

todas las f.n cerradas y tipables son valores ?

En el cálculo-λ^b *Sí*

En el cálculo-λ^{bn} *No*

Las f.n que no son valores se llaman **términos de error**

4.5 Propiedades de la evaluación

- **Determinismo** (Teorema)

  Si $M \to N_1$ *y* $M \to N_2$ *entonces* $N_1 = N_2$

- **Preservación de tipos** (Teorema)

  Si $\vdash M : \tau$ *y* $M \to N$ *entonces* $\vdash N : \tau$

- **Progreso**(Teorema)

  Si $\vdash M : \tau$ *entonces* :

  1. O bien M es un **valor**
  2. O bien existe N tal que $M \to N$

- **Terminación** (Teorema)

  Si $\vdash M : \tau$, *entonces no hay una cadena infinita de pasos* :

  $M \to M_1 \to M_2 \to ...$