Clase Práctica 5 : Cálculo λ

Tomás Felipe Melli

July 10, 2025

Índice

1 Sintaxis

Los tipos del cálculo lambda simplemente tipado con booleanos se define mediante la siguiente gramática :

$\sigma ::= Bool \mid \sigma \to \sigma$ `acá sigma es un metavalor, no un tipo`

y sus términos son los siguientes :

$M ::= x \mid \lambda x : \sigma.M \mid true \mid false \mid if\ M\ then\ M\ else\ M$

`acá M es metavariable y` x `no es ocurrencia de la variable, sólo indica que es el nombre del parámetro de la función`
donde $x \in \mathcal{X}$, el conjunto de todas las variables. Llamamos $\mathcal{T}$ al conjunto de todos los términos.

1.1 Variables Libres y Ligadas

Las **variables libres** son todas aquellas **fuera del alcance de las** λs. Se define la función $fv : \mathcal{T} \to \mathcal{P}(\mathcal{X})$, que dado un término devuelve un conjunto de las variables libres en él. Veamos los ejemplos :

$$\begin{aligned}
fv(x) &= \{x\} \\
fv(\lambda x : \sigma.M) &= fv(M)n\{x\} \\
fv(M\ N) &= fv(M) \cup fv(N) \\
fv(true) &= \emptyset \\
fv(false) &= \emptyset \\
fv(\text{if M then N else O}) &= fv(M) \cup fv(N) \cup fv(O)
\end{aligned}$$

Decimos que un **término es cerrado si no tiene variables libres**. M es cerrado si y sólo si $fv(M) = \emptyset$.

1.2 Asociatividad y Precedencia

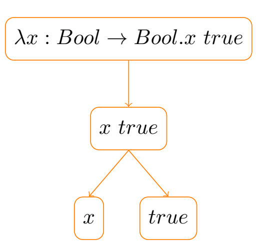
$$\begin{aligned}
\sigma \to \tau \to \rho &= \sigma \to (\tau \to \rho) \neq (\sigma \to \tau) \to \rho \\
M\ N\ O &= (M\ N)\ O \neq M\ (N\ O) \\
\lambda x : \sigma.M\ N &= \lambda x : \sigma.(M\ N) \neq (\lambda x : \sigma.M)\ N
\end{aligned}$$

De lo anterior podemos concluir que :

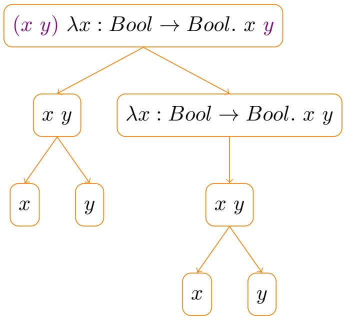
- Las flechas en los tipos asocian a derecha
- La aplicación asocia a izquierda
- El cuerpo de la lambda se extiende hasta el final del término, excepto que haya paréntesis

1.3 Ejercicio : `hallar términos del cálculo lambda`

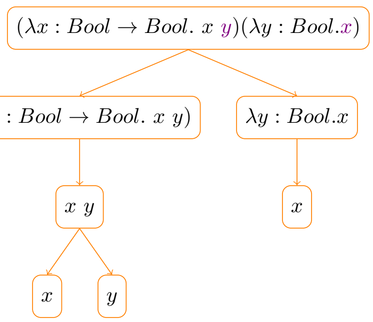
1. $\lambda x : Bool \to Bool.x\ true$



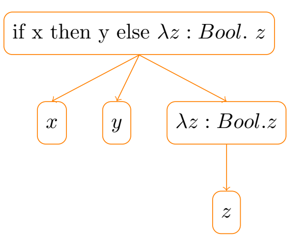
2. $x\ y\ \lambda x : Bool \to Bool.\ x\ y$. Las variables en violeta están libres. Ponemos los paréntisis por claridad.



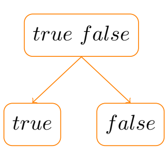
3. $(\lambda x : Bool \to Bool.\ x\ y)(\lambda y : Bool.\ x)$. Marcamos en violeta las libres.



4. Falta término.
5. Falta tipo.
6. if x then y else $\lambda z : Bool.\ z$



7. $\lambda y : \sigma.\ y$. σ **no es un tipo, por tanto, no es un término válido**.
8. $true\ false$



9. $x\ M$. **M no es término** por tanto, $x\ M$ no lo es.
10. if x then $\lambda x : Bool.x$. Falta el `else`

2 Tipado

2.1 Tipos

La gramática que define los tipos del cálculo lambda simplemente tipado con booleanos es

$$\sigma ::= Bool \mid \sigma \to \sigma$$

Los **contextos** son conjuntos finitos de asociaciones entre tipos y variables. Por ejemplo :

$$\Gamma_1 = y : Bool \to Bool$$
$$\Gamma_2 = y : Bool \to Bool, x : Bool$$

Son contextos válidos pero

$$\Gamma_3 = y : Bool \to Bool, y : Bool$$

No lo es, ya que **son conjuntos y no pueden estar repetidos ni haber contradicciones**. Si pensamos en

$$\Gamma_4 = y : Bool \to Bool, y : Bool \to Bool$$

También está mal.

2.2 Juicios de tipado

Un juicio de tipado es la **relación** $\Gamma \vdash M : \tau$ y se lee "en el contexto gamma, M es del tipo tau". Ejemplo :

$$\begin{aligned} \{x : Bool \to Bool\} &\vdash x : Bool \to Bool \\ &\vdash true : Bool \\ \{f : Bool \to Bool, x : Bool\} &\vdash f\ x : Bool \end{aligned}$$

Son juicios de tipado válidos.

2.3 Sistema de tiapdo

Los juicios de tipado válidos se pueden derivar mediante el siguiente sistema de reglas de deducción :

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

$$\frac{}{\Gamma \vdash true : Bool}\ \text{T-True}$$

$$\frac{}{\Gamma \vdash false : Bool}\ \text{T-False}$$

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

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

$$\frac{\Gamma \vdash M : Bool \qquad \Gamma \vdash N_1 : \tau \qquad \Gamma \vdash N_2 : \tau}{\Gamma \vdash \texttt{if M then } N_1 \texttt{ else } N_2 : \tau}\ \text{T-If}$$

La idea del sistema de tipado es darse cuenta que si no podemos aplicar una regla en cada momento, entonces, no tipa.

2.4 Ejercicio : `chequeo de tipos`

1. $\vdash (\lambda x : Bool.\lambda y : Bool.\text{if x then true else y})\ false : Bool \to Bool$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : Bool}\text{T-Var} \qquad \dfrac{}{\Gamma \vdash true : Bool}\text{T-True} \qquad \dfrac{}{\Gamma \vdash y : Bool}\text{T-Var}}{\Gamma = \{x : Bool, y : Bool\} \vdash \text{if x then true else y} : Bool}\text{T-If}}{x : Bool \vdash \lambda y : Bool.\text{if x then true else y} : Bool \to Bool}\text{T-Abs}}{\vdash (\lambda x : Bool.\lambda y : Bool.\text{if x then true else y}) : Bool \to (Bool \to Bool)}\text{T-Abs} \qquad \dfrac{}{\vdash false : Bool}\text{T-False}}{\vdash (\lambda x : Bool.\lambda y : Bool.\text{if x then true else y})\ false : Bool \to Bool}\text{T-App}$$

2. $\{x : Bool\} \vdash true : Bool$

$$\dfrac{}{\{x : Bool\} \vdash true : Bool}\text{T-True}$$

3. $\vdash \text{if x then x else z} : Bool$

$$\dfrac{\vdash x : Bool \qquad \vdash x : Bool \qquad \vdash z : Bool}{\vdash \text{if x then x else z} : Bool}\text{T-If}$$

   No se pueden inferir los tipos de `x` y `z`. **Un término no cerrado no puede ser tipable en el contexto vacío**.

4. $\{x : Bool\} \vdash \text{if x then x else } (\lambda y : Bool.y) : Bool \to Bool$

$$\dfrac{\vdash x : Bool \qquad \vdash x : Bool \to Bool \qquad \vdash \lambda y : Bool.y : Bool \to Bool}{\{x : Bool\} \vdash \text{if x then x else } (\lambda y : Bool.y) : Bool \to Bool}\text{T-If}$$

   El tema es que no puedo aplicar `T-Var` a $x : Bool \to Bool$ entonces no tipa. Además, sólo puedo aplicar la regla `T-If` al principio.

2.5 Ejercicio : `chequeo de tipos con incógnitas`

Nos piden derivar el siguiente juicio de tipado (identificando qué tipos pueden ser τ, σ y ρ)

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : \epsilon \to \alpha}\textbf{(*)T-Var} \qquad \dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : \sigma \to \tau \to \epsilon}\textbf{(*)T-Var} \qquad \dfrac{}{\Gamma \vdash y : \sigma}}{\Gamma \vdash xy : \tau \to \epsilon}\text{T-App} \qquad \dfrac{}{\Gamma \vdash z : \tau}\text{T-Var}}{\Gamma \vdash xyz : \epsilon}\text{T-App}}{\Gamma = \{x : \rho, y : \sigma, z : \tau\} \vdash x\ (xyz) : \alpha}\text{T-App}}{x : \rho, y : \sigma \vdash \lambda z : \tau.x\ (xyz) : \tau \to \alpha}\text{T-Abs}$$

$$\dfrac{\dfrac{x : \rho, y : \sigma \vdash \lambda z : \tau.x\ (xyz) : \tau \to \alpha}{x : \rho \vdash \lambda y : \sigma.\lambda z : \tau.x\ (xyz) : \sigma \to \tau \to \alpha}\text{T-Abs}}{\vdash \lambda x : \rho.\lambda y : \sigma.\lambda z : \tau.x\ (xyz) : \rho \to \sigma \to \tau \to \alpha}\text{T-Abs}$$

(*)$\rho = \epsilon \to \alpha$
(*)$\rho = \sigma \to (\tau \to \epsilon)$

$\Rightarrow \epsilon = \sigma, \alpha = \tau \to \epsilon = \tau \to \sigma$
$\Rightarrow \rho = \sigma \to \tau \to \sigma$

$\vdash \lambda x : \rho.\lambda y : \sigma.\lambda z : \tau.x\ (xyz) : (\sigma \to \tau \to \sigma) \to \sigma \to \tau \to \tau \to \sigma$. Sin embargo, este juicio de tipado no es válido ya que sigue habiendo metavariables. Lo que habría que hacer es poner que $\forall \sigma$ y τ **tipos**.

2.6 Ejercicio : `tipos habitados`

Decimos que un tipo τ está habitado si existe un término M tal que el juicio $\vdash M : \tau$ es derivable. Por ejemplo :

$id_\sigma = \lambda x : \sigma.x$
$\sigma \to \sigma$ `es habitado ya que` $\vdash id_\sigma : \sigma \to \sigma$

1. $\tau \to \sigma \to \tau$
   $\vdash \lambda x : \sigma.\lambda y : \tau.x : \sigma \to \tau \to \sigma \quad \forall \sigma, \tau$ `tipos`
2. $(\tau \to \rho) \to (\sigma \to \tau) \to (\sigma \to \rho)$
   $\vdash \lambda f : \tau \to \sigma.\lambda g : \sigma \to \tau.\lambda x : \sigma.f\ (g\ x) : (\tau \to \rho) \to (\sigma \to \tau) \to (\sigma \to \rho) \quad \forall \sigma, \tau, \rho$ `tipos`

3 Semántica operacional

Consiste en un **conjunto de reglas que definen la relación de reducción entre términos**. Cuando $M \to N$ decimos que M reduce o reescribe a N.

3.1 Formas Normales

Un término es o está en forma normal cuando **no existe ninguna relga que lo reduzca a otro**

3.2 Determinismo

Decimos que la reducción está determinada cuando **cada término que no está en forma normal tiene una única forma de reducir**.

3.3 Estrategias de reducción

Para implementar un lenguaje, necesitamos una relación de reducción que esté determinada. Existen estrategias **call-by-name** y **call-by-value**. Nos vamos a enfocar en **call-by-value** que es básicamente aplicar una función a algo cuando ese algo ya es un valor (meterlo en el cuerpo de la lambda cuando ya es valor). Nos vamos a enfocar en matener el determinismo en las reglas de reducción.
La siguiente gramática de valores y reglas de reducción definen la estrategia **call-by-value**

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

Importante : tipan en el contexto vacío, no tienen variables libres. Si un término es un valor entonces es una forma normal, pero no vale al revés. Ya que la f.n no se puede reducir pero no quiere decir que tipe. Un valor no puede tener variables libres.

$(\lambda x : \sigma.M)\ V \to M\{x := V\}$	$(E - AppAbs\ o\ \beta)$
`if true then M else N` $\to M$	$(E - IfTrue)$
`if true then M else N` $\to N$	$(E - IfFalse)$

Si $M \to N$, entonces :

$M\ O \to N\ O$	$(E - App1\ o\ \mu)$
$V\ M \to V\ N$	$(E - App1\ o\ \nu)$
`if M then O else P` $\to$ `if N then O else P`	$(E - If)$

3.4 Valores

Los valores son los resultados esperados de los programas. Se definen como **términos cerrados y bien tipados V producidos por la gramática de valores**. Deben tipar en el contexto vacío y no se pueden reducir. Importante lo de que sean cerrados.

3.4.1 Ejercicio : `decidir cuáles términos son valores`

1. if true then $(\lambda x : Bool.x)$ else $(\lambda x : Bool.false)$
   `No es valor ya que podemos aplicar E-IfTrue para reducirlo`
2. $\lambda x : Bool.false$
   `Valor, es una lambda`
3. $(\lambda x : Bool.x)\ false$
   `No es valor, es una aplicación. Podemos aplicar E-AppAbs o` β
4. true
   `Es valor`
5. if x then true else false
   `No es valor porque no es cerrado`
6. $\lambda x : Bool.(\lambda y : Bool.x)\ false$
   `Es valor ya que no se puede reducir y tipa en el contexto vacío. La lambda es un valor y false es un valor`

7. $\lambda x : Bool \to Bool.x\ true$
   No es valor

3.4.2 Ejercicio : evaluar expresiones (paso a paso)

1. $((\lambda x : Bool.\lambda y : Bool.\text{if x then true else y})\ false)\ true$

$$\overbrace{((\lambda x : Bool.\lambda y : Bool.\text{if x then true else y})\ false)}^{M}\ \overbrace{true}^{N}$$

$$\xrightarrow{E-AppAbs,\ E-App1} \overbrace{(\lambda x : Bool.\lambda y : Bool.\text{if x then true else y})}^{Valor,Lambda}\ \overbrace{true}^{Valor}$$

$$\xrightarrow{E-AppAbs} \text{if false then true else true}$$

Aplico esto porque tengo una aplicación M N (no valor E-App1) "Aplico N a la reducción de M" y E-App

$$\xrightarrow{E-IfFalse} true$$

Aplico esto porque tengo una lambda aplicada a un valor

2. $(\lambda x : Bool.\lambda y : Bool \to Bool.y(yx))((\lambda z : Bool.true)\ false)(\lambda w : Bool.w)$
   Miramos los paréntesis y separamos

$$\overbrace{\underbrace{(\lambda x : Bool.\lambda y : Bool \to Bool.y(yx))}_{M}\underbrace{((\lambda z : Bool.true)\ false)}_{N}}^{M'}\overbrace{(\lambda w : Bool.w)}^{N'}$$

$$\xrightarrow{E-App1,\ E-App2,\ E-AppAbs} \overbrace{(\underbrace{(\lambda x : Bool.\lambda y : Bool \to Bool.y(yx))}_{M'}\ \underbrace{true}_{N'})}^{M}\overbrace{(\lambda w : Bool.w)}^{N}$$

E-App1 : porque tenemos M' N' y reducimos M' ya que no es valor

E-App2 : porque tenemos M que es valor y tenemos algo de la forma V N y reducimos N,
como consecuencia reducimos a M'

E-AppAbs : para reducir N porque tengo lambda aplicada a un valor.

Siempre aplicamos un E-AppAbs a la vez

$$\xrightarrow{E-App1,\ E-AppAbs} (\lambda y : Bool \to Bool.y\ (y\ true))\ (\lambda w : Bool.w)$$

E-App1 : porque tengo M N y M no es valor, entonces aplicamos beta

E-AppAbs : para reducir M que es lambda aplicada a un valor

$$\xrightarrow{E-AppAbs} \underbrace{(\lambda w : Bool.w)}_{Valor\ :\ lambda}\underbrace{((\lambda w : Bool.w)\ true)}_{N}$$

E-AppAbs : ya que tengo una lambda aplicada a un valor

$$\xrightarrow{E-App2,\ E-AppAbs} (\lambda w : Bool.w)\ true$$

E-App2 : porque tengo lambda aplicada a N, no es valor y por tanto aplico lambda (reducción de N)

E-AppAbs : para reducir , que es una lambda aplicada a un valor

$$\xrightarrow{E-AppAbs} true$$

E-AppAbs : para reducir una lambda aplicada a un valor

4 Determinismo

Es útil para ejercicios en los que se agregan reglas. Vamos a hacer inducción sobre la cantidad de pasos de la derivación. Los casos bases van a usar E-IfTrue, E-IfFalse y E-AppAbs. Los pasos inductivos van a usar E-If, E-App1 y E-App2, o sea las reglas que tienen premisas.

4.1 Ejercicio : `Probar que la semántica operacional de cálculo lambda con booleanos, con la estrategia call-by-value está determinada.`

O sea, queremos probar que si $M \to M_1$ y $M \to M_2$, entonces $M_1 = M_2$

Casos base

- $M \to M_1$ con la regla E-IfTrue:
  - $M =$ if true then M_a else M_b
  - $M \to M_a$ por E-IfTrue, con $M_a = M_1$
  - No puedo aplicar E-AppAbs porque no hay λ, ni E-App1 ni E-App2 porque no hay dos términos. Tampoco E-IfFalse porque la guarda no es `false`, ni E-If porque la guarda ya es un valor.
  - Luego, si $M \to M_2$, lo hace con la regla E-IfTrue. Entonces $M_2 = M_a = M_1$.
- $M \to M_1$ con la regla E-IfFalse:
  - $M =$ if false then M_a else M_b
  - $M \to M_b$ por E-IfFalse, con $M_b = M_1$
  - No puedo aplicar E-AppAbs, E-App1 ni E-App2, ni tampoco E-IfTrue ni E-If porque la guarda es un valor distinto de `true`.
  - Luego, si $M \to M_2$, lo hace con la regla E-IfFalse. Entonces $M_2 = M_b = M_1$.
- $M \to M_1$ con la regla E-AppAbs:
  - $M = (\lambda x : \sigma.M)\ V$
  - $M \to M\{x := V\}$ por E-AppAbs, con $M\{x := V\} = M_1$
  - No puedo aplicar E-App1 ni E-App2 porque ambos términos son valores. Tampoco E-If, E-IfFalse, o E-IfTrue porque no hay una estructura `if then else`.
  - Luego, si $M \to M_2$, lo hace con la regla E-AppAbs. Entonces $M_2 = M\{x := V\} = M_1$.

Pasos inductivos

- $M \to M_1$ con la regla E-App1:
  - $M = M_a\ M_b$ y $M \to M'_a\ M_b = M_1$, siendo $M_a \to M'_a$
  - No puedo aplicar E-AppAbs ni E-App2 porque M_a no es un valor. Tampoco E-If, E-IfTrue ni E-IfFalse porque no tengo una estructura `if then else`.
  - Luego, si $M \to M_2$, lo hace con la regla E-App1. Entonces $M_2 = M''_a\ M_b$, con $M_a \to M''_a$
  - Por hipótesis inductiva, si $M_a \to M'_a$ y $M_a \to M''_a$, entonces $M'_a = M''_a$
  - Luego, $M_2 = M''_a\ M_b = M'_a\ M_b = M_1$
- $M \to M_1$ con la regla E-App2:
  - $M = V\ M_b$ y $M \to V\ M'_b = M_1$, siendo $M_b \to M'_b$
  - No puedo aplicar E-AppAbs ni E-App1 porque M_b no es un valor. Tampoco E-If, E-IfTrue ni E-IfFalse porque no tengo una estructura `if then else`.
  - Luego, si $M \to M_2$, lo hace con la regla E-App2. Entonces $M_2 = V\ M''_b$, con $M_b \to M''_b$
  - Por hipótesis inductiva, si $M_b \to M'_b$ y $M_b \to M''_b$, entonces $M'_b = M''_b$
  - Luego, $M_2 = V\ M''_b = V\ M'_b = M_1$
- $M \to M_1$ con la regla E-If:
  - $M =$ if N then P else Q y $M \to$ if N' then P else $Q = M_1$, siendo $N \to N'$
  - No puedo aplicar E-AppAbs, E-App1, ni E-App2 porque M no es una aplicación. Tampoco E-IfTrue ni E-IfFalse porque la guarda no es un valor booleano.
  - Luego, si $M \to M_2$, lo hace con la regla E-If. Entonces $M_2 =$ if N'' then P else Q, con $N \to N''$
  - Por hipótesis inductiva, si $N \to N'$ y $N \to N''$, entonces $N' = N''$
  - Luego, $M_2 =$ if N'' then P else $Q =$ if N' then P else $Q = M_1$

5 Extensión con números naturales

5.1 Sintaxis y Tipado

Se extienden las gramáticas de términos y tipos de la siguiente manera:

$$\sigma ::= \ldots \mid Nat$$
$$M ::= \ldots \mid zero \mid succ(M) \mid pred(M) \mid isZero(M)$$

Se extiende el sistema de tipado con las siguientes reglas :

$$\frac{}{\Gamma \vdash zero : Nat}\ \text{T-Zero}$$

$$\frac{\Gamma \vdash M : Nat}{\Gamma \vdash pred(M) : Nat}\ \text{T-Pred}$$

$$\frac{\Gamma \vdash M : Nat}{\Gamma \vdash succ(M) : Nat}\ \text{T-Succ}$$

$$\frac{\Gamma \vdash M : Nat}{\Gamma \vdash isZero(M) : Bool}\ \text{T-isZero}$$

5.2 Semántica operacional

Se extienden los valores como sigue :

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

Además, usamos la notación $\underline{n}$ para $succ^n(zero)$ con $n \geq 0$.
Se extiende la semántica operacional con las siguientes reglas

$\texttt{pred(succ(V))} \to \texttt{V}$	(E-PredSucc)
$\texttt{isZero(zero)} \to \texttt{true}$	(E-isZero$_0$)
$\texttt{isZero(succ(V))} \to \texttt{false}$	(E-isZero$_n$)

Si $M \to N$, entonces :

$\texttt{succ(M)} \to \texttt{succ(N)}$	(E-Succ)
$\texttt{pred(M)} \to \texttt{pred(N)}$	(E-Pred)
$\texttt{isZero(M)} \to \texttt{isZero(N)}$	(E-isZero)

5.3 Ejercicio : `extensión`

Esta extensión matiene las propiedades de determinismo, preservación de tipos y progreso ?

No mantiene el progreso por `pred(zero)`.

Qué términos representan las expresiones $\underline{0}, \underline{1}, \underline{2}$? Cómo reducen ?

Representan `zero, succ(zero)` y `succ(succ(zero))`. No reducen, ya son valores.

Demostrar los siguientes juicios de tipado

- $\vdash (\lambda x : Nat.succ(x))zero : Nat$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : Nat \vdash x : Nat}\ \text{T-Var}}{x : Nat \vdash succ(x) : Nat}\ \text{T-Succ}}{\vdash (\lambda x : Nat.succ(x)) : Nat \to Nat}\ \text{T-Abs} \qquad \dfrac{}{\vdash zero : Nat}\ \text{T-Zero}}{\vdash (\lambda x : Nat.succ(x))zero : Nat}\ \text{E-App}$$

Escribir la reducción paso por paso de los siguientes términos

- $isZero(succ(pred(succ(zero))))$

  $\texttt{isZero}\,\underbrace{(succ(\overbrace{pred(succ(zero))}^{M'}))}_{M}$

  $\overset{E-IsZero,\ E-Succ,\ E-PredSucc}{\rightarrow}$ `isZero(succ(zero))`

  `E-IsZero :  porque tenemos isZero de M y entonces aplicamos isZero de la reducción de M (que existe porque M no es valor ya que es succ de algo que reduce)`

  `E-Succ :  para reducir M, que es succ(M’) que reduce a succ de la reducción de M’`

  `E-PredSucc :  para reducir M’`

  $\overset{E-IsZero_n}{\rightarrow}$ `false`

- $isZero(succ(pred(succ(zero))))$. No reduce a un valor.
- $isZero(succ(pred(succ(zero))))$. No reduce a un valor.
- $isZero(pred(succ(x)))$. Tiene variables libres, no reducimos.

Si queremos recuperar la propiedad de progreso, agregamos esta regla (cambia la semántica)

$$\texttt{pred(zero)} \rightarrow \texttt{zero} \qquad \texttt{(E-Pred)}_0$$

5.4 Simplificando la escritura : `macros`

Para expresiones que usamos con frecuencia podemos definir macros como

$\texttt{Id}_\tau \overset{Def}{=} \lambda x : \tau.x$

$\texttt{and} \overset{Def}{=} \lambda x : Bool.\lambda y : Bool.\texttt{if x then y else false}$

5.5 Cambiando las reglas semánticas

Supongamos que agregamos la siguiente regla para las abstracciones :

$M \rightarrow N,$ `entonces:`

$$\lambda x : \tau.M \rightarrow \lambda x : \tau.N \qquad (\psi)$$

5.5.1 Ejercicio : `modificación - determinismo - reducción`

Repensar el conjunto de valores para respetar esta modificación, pensar por ejemplo si $\lambda x : Bool.Id_{Bool}\ true$ es o no un valor. Y $\lambda x : Bool.x$?

$V ::= true \mid false \mid \lambda x : \sigma.F \mid 0 \mid succ(V)$ `donde F es una forma normal`

Qué reglas deberían modificarse para no perder el determinismo ?

$(\lambda x : Bool.\text{if true then false else true})\ true$

$\overbrace{\overset{\beta}{\rightarrow} \text{if true then false else true}}^{opción1}$

$\overbrace{\overset{\psi,\ E-App1,\ E-IfTrue}{\rightarrow} (\lambda x : Bool.false)\ true}^{opción2}$

Dado que podemos reducir un término a 2 diferentes, **perdemos determinismo**

$(\lambda x : \sigma.F)V \rightarrow F\{x := V\} \quad (\beta')$

Utilizando la nueva regla y los valores definidos, reducir la siguiente expresión

$\lambda z : Nat \to Nat.(\lambda x : Nat \to Nat.x\ \underline{23})\lambda z : Nat.\underline{0}$

$\lambda z : Nat \to Nat.\overbrace{((\lambda x : Nat \to Nat.x\ \underline{23})(\lambda z : Nat.\underline{0}))}^{M}$

$\xrightarrow{\psi,\ \beta} \lambda z : Nat \to Nat.(\lambda x : Nat \to Nat.x\ \underline{23})$

ψ : `ya que tengo` $\lambda z : Nat \to Nat.M$ `donde M no es una forma normal, por tanto la reducimos`

β : `para reducir M porque` $\underline{23}$ `es una forma normal`

$\xrightarrow{\psi,\ \beta'} \lambda z : Nat \to Nat.\underline{0}$

ψ : `análogo a lo anterior, reducimos M`

β' : `para reducir M, que es` $\lambda z : Nat.M'$ `(donde M’ está en forma normal) aplicada a un valor`

Podemos concluir que la regla no suma nada.