# Práctica 4 : Cálculo $\lambda$: Tipado y Semántica Operacional

Tomás Felipe Melli

May 8, 2025

## Índice

# 1 Sintaxis

## 1.1 Ejercicio 1

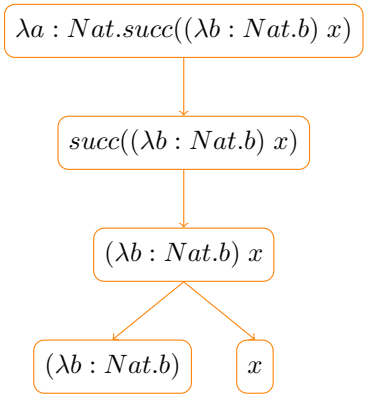
Queremos ver si los siguientes términos están **sintácticamente** bien (pueden ser generados con las grámaticas que conocemos) y a qué categoría pertenecen.

1. $x$ : $x$ está bien formado y responde a la categoría de *variable* dentro de nuestro esquema conocido de construcción,
2. $x\ x$: está bien formado sintácticamente y responde a la categoría de *aplicación*.
3. $M$ : está mal formado, M es variable de tipo.
4. $M\ M$: está bien formado y categoriza a la *aplicación*.
5. $true\ false$ : está bien formado y categoriza a *aplicación*.
6. $true\ succ(false\ true)$ : está bien formado y categorizarí a una aplicación del tipo $M\ N$ con $M = true$ y $N = succ(O)$ con $O = false\ true$ (*aplicación*)
7. $\lambda x.isZero(x)$: mal formado, representaría una *abstracción*, pero le falta el tipo de $x$.
8. $\lambda x : \sigma.succ(x)$: mal formada, $\sigma$ es variable de tipo.
9. $\lambda x : Bool.succ(x)$ : bien formada la *abstracción*.
10. $\lambda x :$ if $true$ then$Bool$ else $Nat.x$: mal formado.
11. $\sigma$: mal formado, es una variable de tipo.
12. $Bool$ : es un tipo, está bien.
13. $Bool -> Bool$ : es un tipo, está bien.
14. $Bool -> Bool -> Nat$ : es un tipo, está bien.
15. $(Bool -> Bool) -> Nat$ : es un tipo, está bien.
16. $succ\ true$ : mal formado.
17. $\lambda x : Bool.$ if $zero$ then $true$ else $zero\ succ\ true$. Bien formada la *abstracción*.

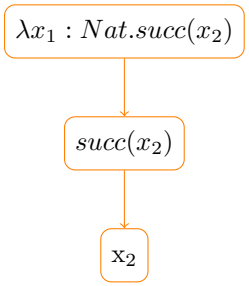
## 1.2 Ejercicio 3

Cuando nos piden marcar **ocurrencias** se refiere a la *aparición de la variable en el árbol sintáctico.*

1. Marcar las ocurrencias del término $x$ como subtérmino en $\lambda x : Nat.succ((\lambda x : Nat.x)\ x)$
   Veamos el árbol : le vamos a cambiar los nombres por practicidad




2. Ocurre $x_1$ como subtérmino de $\lambda x_1 : Nat.succ(x_2)$

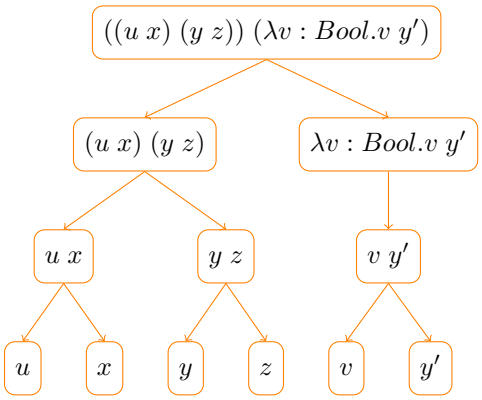


3. Ocurre $x\ (y\ z)$ como subtérmino en $u\ x\ (y\ z)$? La **aplicación asocia a izquierda** por tanto, $(u\ x)\ (y\ z)$ y como consecuencia, no ocurre como subtérmino.

## 1.3 Ejercicio 4

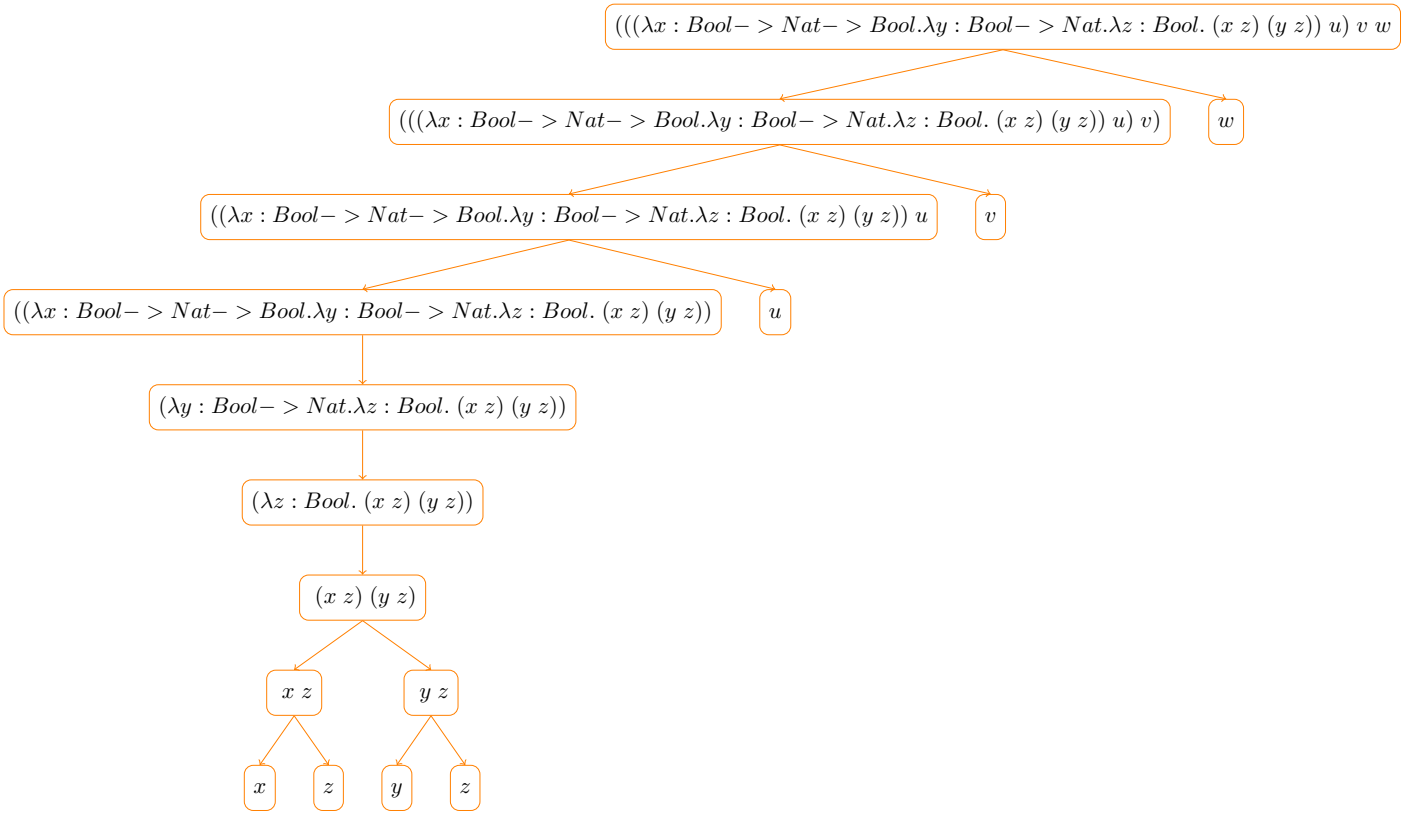
Para los siguientes términos nos piden, **poner los paréntesis de acuerdo a la convención**, **realizar el árbol sintáctico e indicar la ocurrencias libres y ligadas** y en cúales aparece la expresión $(\lambda x : Bool- > Nat- > Bool.\lambda y : Bool- > Nat.\lambda z : Bool.\ x\ z\ (y\ z)\ )\ u$ como subtérmino.

1. $u\ x\ (y\ z)\ (\lambda v : Bool.v\ y) \Rightarrow ((u\ x)\ (y\ z))\ (\lambda v : Bool.v\ y)$
   Renombramos y armamos el árbol



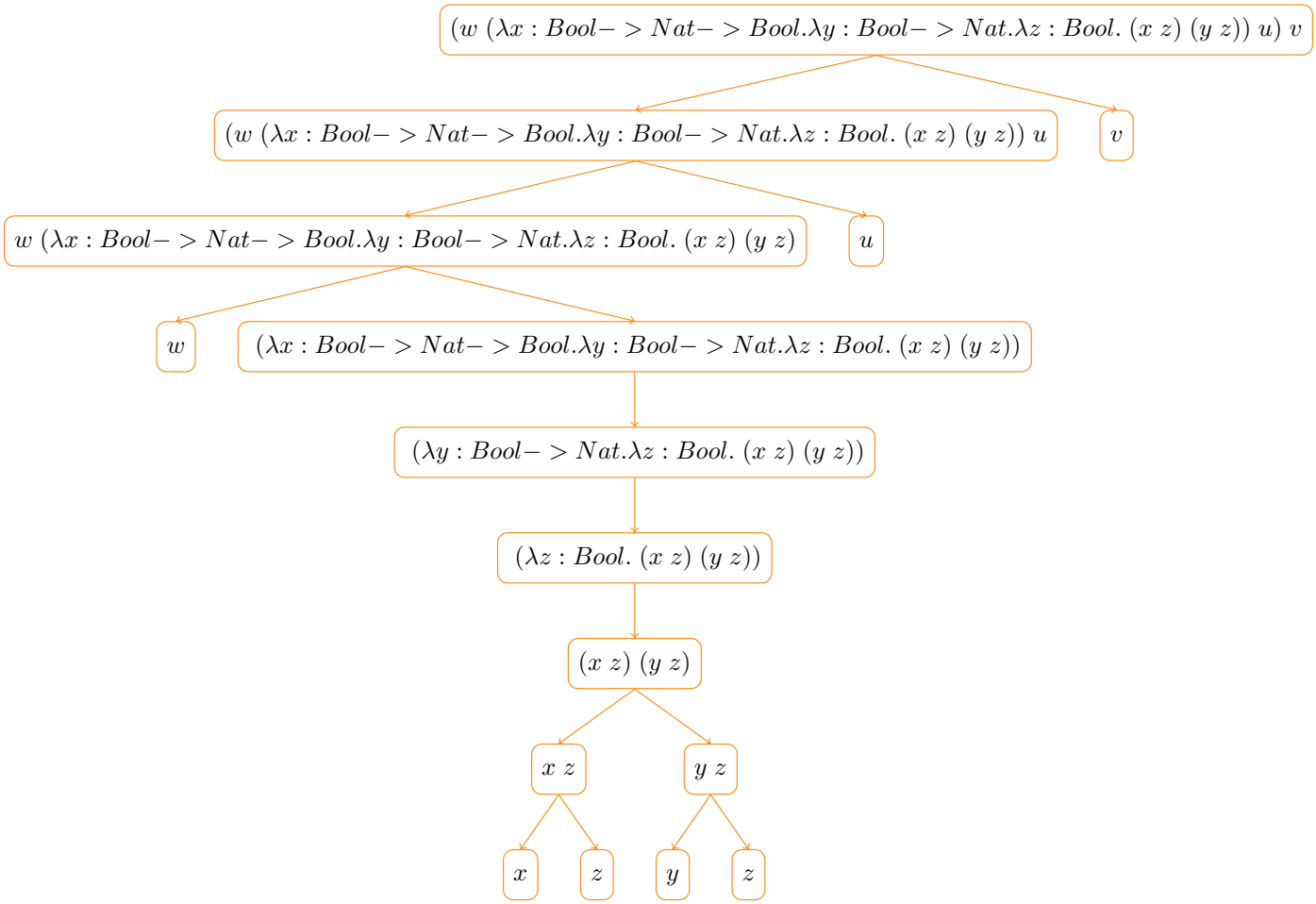

   El conjunto de las variables libres es $\{y'\}$, luego, todas ligadas. No aparece la expresión.

2. $(\lambda x : Bool- > Nat- > Bool.\lambda y : Bool- > Nat.\lambda z : Bool.\ x\ z\ (y\ z))\ u\ v\ w \Rightarrow$
   $(((\lambda x : Bool- > Nat- > Bool.\lambda y : Bool- > Nat.\lambda z : Bool.\ (x\ z)\ (y\ z))\ u)\ v)\ w$. Veamos el árbol




   Ahora bien, las variables libres son $\{u, v, w\}$, las demás están ligadas. En esta expresión sí es subtérmino.

3. $w\ (\lambda x : Bool -> Nat -> Bool.\lambda y : Bool -> Nat.\lambda z : Bool.\ x\ z)\ (y\ z))\ u\ v \Rightarrow$
$(w\ (\lambda x : Bool -> Nat -> Bool.\lambda y : Bool -> Nat.\lambda z : Bool.\ (x\ z)\ (y\ z))\ u)\ v$
Miremos el árbol




En este caso, también aparece como subtérmino la expresión. El conjunto de variables libres es $\{w, u, v\}$.

# 2 Tipado

## 2.1 Ejercicio 6 : Derivaciones

Nos piden dar una derivación, y en caso de no poder dar explicación de por qué, para cada unos de los siguientes juicios de tipado:

1. $\vdash if\ true\ then\ zero\ else\ succ(zero)\ : Nat$

$$\dfrac{\dfrac{}{\Gamma \vdash true : Bool}\ \text{ax} \quad \dfrac{}{\Gamma \vdash zero : Nat}\ \text{ax} \quad \dfrac{\dfrac{}{\Gamma \vdash zero : Nat}\ \text{ax}}{\Gamma \vdash succ(zero) : Nat}\ \text{T-SUCC}}{\vdash if\ true\ then\ zero\ else\ succ(zero)\ : Nat}\ \text{T-IF}$$

2. $x : Nat, y : Bool \vdash if\ true\ then\ false\ else\ (\lambda z : Bool.z)\ true : Bool.$

$$\dfrac{\dfrac{}{\vdash true : Bool}\ \text{ax} \quad \dfrac{}{\vdash false : Bool}\ \text{ax} \quad \dfrac{\dfrac{\dfrac{}{z : Bool \vdash z : Bool}\ \text{ax}}{\vdash (\lambda z : Bool.z) : Bool -> Bool}\ \text{T-ABS} \quad \dfrac{}{\vdash true : Bool}\ \text{ax}}{\vdash (\lambda z : Bool.z)\ true : Bool}\ \text{T-APP}}{x : Nat, y : Bool \vdash if\ true\ then\ false\ else\ (\lambda z : Bool.z)\ true : Bool}\ \text{T-IF}$$

3. $\vdash if\ \lambda x : Bool\ then\ zero\ else\ succ(zero) : Nat$

$$\dfrac{\dfrac{\text{No cumple con la regla T-IF}}{\vdash (\lambda x : Bool.x) : Bool \to Bool} \quad \dfrac{}{\vdash zero : Nat}\ \text{ax} \quad \dfrac{\ldots}{\vdash succ(zero) : Nat}}{\vdash if\ \lambda x : Bool.x\ then\ zero\ else\ succ(zero) : Nat}\ \text{T-IF}$$

4. $x : Bool -> Nat, y : Bool \vdash x\ y : Nat$

$$\frac{x : Bool -> Nat, y : Bool \vdash x\ : Bool -> Nat \qquad x : Bool -> Nat, y : Bool \vdash y : Nat}{x : Bool -> Nat, y : Bool \vdash x\ y : Nat}\ \text{T-APP}$$

## 2.2 Ejericio 7

Se modifica la regla de tipado de la abstracción ($\rightarrow_i$) y se la cambia por

$$\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \lambda x : \sigma.M : \sigma \rightarrow \tau}\ \rightarrow_{i2}$$

Con esto en mente, nos piden exhibir un juicio de tipado derivable en el original, pero no en este. Veamos

$$\frac{\Gamma \vdash z : Bool}{\vdash (\lambda z : Bool.z) : Bool -> Bool}\ \rightarrow_{i2}$$

Pero como vimos en el ejercicio 6 punto 2, con la regla tradicional, es derivable. El problema es que no podemos deducir con el conjunto de premisas que $z : Bool$ y por tanto, no es derivable en $\rightarrow_{i2}$

## 2.3 Ejericio 9 : Tipos habitados

Decimos que un tipo $\tau$ está **habitado, si existe un término M tal que el juicio** $\vdash M : \tau$ **es derivable**. En este caso, decimos que **M es un habitante de** $\tau$. Por ejemplo, dado un tipo $\sigma$, la identidad $\lambda x : \sigma.x$ es un habitante del tipo $\sigma \rightarrow \sigma$. Demostrar que los siguientes tipos están habitados (para cualquier $\sigma, \tau\ y\ \rho$ ):

1. $\sigma \rightarrow \tau \rightarrow \sigma$
   Queremos algo que cumpla

   $$\frac{\ldots}{\vdash M : \sigma \rightarrow \tau \rightarrow \sigma}$$

   Con esto en mente, nos proponemos llegar a algo como

   $$\dfrac{\dfrac{\dfrac{}{\ldots \vdash M'' : \sigma}}{\ldots \vdash M' : \tau \rightarrow \sigma}\ \text{T-ABS}}{\ldots \vdash M : \sigma \rightarrow \tau \rightarrow \sigma}\ \text{T-ABS}$$

   Con esto, podríamos pensar qué $\lambda$ armar y qué premisas son necesarias para que cumpla. Arranquemos por premisas

   $$\dfrac{\dfrac{\dfrac{}{x : \sigma, y : \tau \vdash M'' : \sigma}\ \text{T-ABS}}{x : \sigma \vdash M' : \tau \rightarrow \sigma}\ \text{T-ABS}}{\vdash M : \sigma \rightarrow \tau \rightarrow \sigma}$$

   Ahora completamos las expresiones M

   $$\dfrac{\dfrac{\dfrac{}{x : \sigma, y : \tau \vdash x : \sigma}\ \text{ax}}{x : \sigma \vdash \lambda y : \tau.x : \tau \rightarrow \sigma}\ \text{T-ABS}}{\vdash \lambda x : \sigma.\lambda y : \tau.x : \sigma \rightarrow \tau \rightarrow \sigma}\ \text{T-ABS}$$

2. $(\sigma \rightarrow \tau \rightarrow \rho) \rightarrow (\sigma \rightarrow \tau) \rightarrow \sigma \rightarrow \rho$ Qué es este bicho ? Si prestamos atención tenemos una función que recibe un $\sigma\ y\ \tau$ otra que recibe $\sigma$ y devuelve un $\tau$ y otra que recibe un $\sigma$ y devuelve $\rho$. Les ponemos nombres :

   $$f :: \sigma \rightarrow \tau \rightarrow \rho$$
   $$g :: \sigma \rightarrow \tau$$
   $$x :: \sigma$$
   $$h :: f\ x\ (g\ x)$$

   Con esta idea en mente...

$$\frac{\dots}{\vdash M : (\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho}$$

Quién es M ?

$$\dfrac{\dfrac{\dfrac{\dfrac{\dots}{f : \sigma \to \tau \to \rho, g : \sigma \to \tau, x : \sigma \vdash f\ x(g\ x) : \rho}\ \text{T-ABS}}{f : \sigma \to \tau \to \rho, g : \sigma \to \tau \vdash \lambda x : \sigma. f\ x(g\ x) : \sigma \to \rho}\ \text{T-ABS}}{f : \sigma \to \tau \to \rho \vdash \lambda g : (\sigma \to \tau).\lambda x : \sigma. f\ x(g\ x) : (\sigma \to \tau) \to \sigma \to \rho}\ \text{T-ABS}}{\vdash \lambda f : (\sigma \to \tau \to \rho).\lambda g : (\sigma \to \tau).\lambda x : \sigma.\ f\ x(g\ x) : (\sigma \to \tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho}\ \text{T-ABS}$$

3. $(\sigma \to \tau \to \rho) \to \tau \to \sigma \to \rho$
4. $(\tau \to \rho) \to (\sigma \to \tau) \to \sigma \to \rho$

## 2.4 Ejericicio 10

Determinar qué tipos representan $\sigma$ y $\tau$ en los siguientes juicios.

1. $x : \sigma \vdash isZero(succ(x)) : \tau$ Se ve rápidamente, pero hagamoslo formal

$$\dfrac{\dfrac{x : \sigma \vdash x : Nat}{x : \sigma \vdash succ(x) : Nat}\ \text{T-SUCC}}{x : \sigma \vdash isZero(succ(x)) : Bool}\ \text{T-ISZERO}$$

2. $\vdash (\lambda x : \sigma.x)\ (\lambda y : Bool.zero) : \sigma$

$$\dfrac{\dfrac{\dfrac{\overline{\sigma\ es\ Nat}}{x : \sigma \vdash x : \tau}}{\vdash (\lambda x : \sigma.x) : \tau \to \sigma}\ \text{T-ABS} \qquad \dfrac{\dfrac{\overline{\quad}}{y : Bool \vdash zero : Nat}\ \text{ax}}{y : Bool \vdash zero : \tau}\ \text{resolvemos a tipo Nat}}{\vdash (\lambda y : Bool.zero) : \tau}\ \text{T-ABS}}{\vdash (\lambda x : \sigma.x)\ (\lambda y : Bool.zero) : \sigma}\ \text{T-APP}$$

3. $y : \tau \vdash if\ (\lambda x : \sigma.x)\ then\ y\ else\ succ(zero)\ : \sigma$
   Sabemos que por **T-IF** que las guardas son siempre del mismo tipo, por tanto, podemos a priori decir que $\tau$ es $Nat$. También por esta regla sabemos que la condición deber ser $Bool$ y concluimos que $\sigma$ responde a este tipo
4. $x : \sigma \vdash x\ y : \tau$

$$\dfrac{\dfrac{\overline{\quad}}{x : \sigma \vdash x : \rho \to \tau} \qquad \dfrac{\overline{\quad}}{x : \sigma \vdash y : \rho}}{x : \sigma \vdash x\ y : \tau}\ \text{T-APP}$$

   Me da la sensación que no se puede porque no cumple con la regla de **T-APP**
5. $x : \sigma, y : \tau \vdash x\ y : \tau$ pasa lo mismo acá.
6. $x : \sigma \vdash x\ true : \tau$ no podemos aplicar x porque no cumple la regla
7. $x : \sigma \vdash x\ true : \sigma$ no podemos aplicar x porque no cumple la regla
8. $x : \sigma \vdash x\ x : \sigma$ no podemos aplicar x porque no cumple la regla

# 3 Semántica

## 3.1 Ejercicio 13

Sean $\sigma$, $\tau$, $\rho$ tipos. Renombrar variables en ambos términos para que las sustituciones no cambien su significado.Según la definición de sustitución, calcular:

1. $(\lambda y : \sigma.\ x\ (\lambda x : \tau.\ x))\{x := (\lambda y : \rho.\ x\ y)\}$
   - Primero, como recomienda la consigna, renombramos
     $$(\lambda y : \sigma.\ x\ (\lambda a : \tau.\ a))\{x := (\lambda y : \rho.\ x\ y)\}$$
   - Luego, por definición de sustitución : reemplazamos cada ocurrencia libre de $x$ en M por N
     $$(\lambda y : \sigma.\ (\lambda y : \rho.\ x\ y)\ (\lambda a : \tau.\ a))$$
2. $(y\ (\lambda v : \sigma.\ x\ v))\{x := (\lambda y : \tau.\ v\ y)\}$
   - Análogo a lo anterior, renombramos primero
     $$(y\ (\lambda v : \sigma.\ x\ v))\{x := (\lambda a : \tau.\ b\ a)\}$$
   - A continuación, seguimos la definición :
     $$(y\ (\lambda v : \sigma.\ (\lambda a : \tau.\ b\ a)\ v))$$

## 3.2 Ejercicio 15

Determinar si las siguientes expresiones son valores

1. $(\lambda x : Bool.x)\ true$
   No es valor ya que se puede evaluar $(\lambda x : Bool.x)\ true \to true$
2. $\lambda x : Bool.\underline{2}$. Es valor
3. $\lambda x : Bool.pred(\underline{2})$. Es valor
4. $\lambda y : Nat.(\lambda x : Bool.pred(\underline{2}))\ true$. Tiene la forma $\lambda y : Nat.M$ así que a priori parecería estar bien
5. $x$ no lo es
6. $succ(succ(zero))$ es valor.

## 3.3 Ejercicio 16

En este ejercicio no hay que considerar la regla $pred(Zero) \to Zero$.
Antes de comenzar miremos las definiciones de

- **Programa** : decimos que M es un programa cueando cumple ser un **término cerrado y tipable**. Es decir, todas sus variables están ligadas, como consecuencia $fv(M) = \emptyset$ y en un contexto $\Gamma$ de tipos, M puede ser probado y derivado y tiene un tipo válido.
- **Forma Normal** : decimos que una forma normal es un **programa M** tal que **no existe M' que cumpla** $M \to M'$

Con esto en mente, nos piden determinar si las siguientes expresiones son programas, en caso de serlo, hacer la evaluación y concluir algo sobre esa evaluación (si es un valor o un error)

1. $(\lambda x : Bool.x)\ true) \xrightarrow{\beta} true$ la regla también se llama *E-APPABS*. Entoces, es un programa, ya que no tiene variables libre y tipa. Lo podemos evaluar con la regla de *evaluación de abstracción* $\beta$ y concluimos que el resultado es un valor.
2. $\lambda x : Nat.pred(succ(x)) \overset{E-PREDSUCC}{\longrightarrow} \lambda x : Nat.x$ por tanto se trata de un programa que evalúa a un término de nuestra gramática. Por tanto, el resultado de la evaluación es un valor.
3. $\lambda x : Nat.pred(succ(y))$ podríamos querer nuevamente aplicar *E-PREDSUCC* pero desafortunadamente esta expresión no es un programa ya que $fv(...) = y$ y por tanto no es cerrado.
4. $(\lambda x : Bool.pred(isZero(x)))\ true)$ en este caso, si quisiésemos aplicar la regla *T-PRED* nos daríamos cuenta que la expresión no tipa, esto se debe a que pred espera un Nat. Como conclusión, la expresión es cerrada, pero no tipa.

5. $(\lambda f : Nat \to Bool.f\ zero)(\lambda x : Nat.isZero(x))$ miramos las expresiones y vemos que ambas cumplen con las reglas de tipado y que además el término es cerrado. Concluimos que se trata de un programa y que procedemos a evaluarlo.
$(\lambda f : Nat \to Bool.f\ zero)(\lambda x : Nat.isZero(x)) \xrightarrow{\beta} (\lambda x : Nat.isZero(x))\ zero \xrightarrow{\beta} isZero(zero) \xrightarrow{E-ISZEROZERO} true$
Conclusión, obtenemos un término de la gramática que fue resultado de una evaluación y por tanto, obtuvimos un valor

6. $(\lambda f : Nat \to Bool.x)(\lambda x : Nat.isZero(x))$ el problema de esta expresión es que la x en $(\lambda f : Nat \to Bool.x)$ está libre y por tanto no puede ser programa un término con $fv(...) \neq \emptyset$.

7. $(\lambda f : Nat \to Bool.f\ pred(zero))(\lambda x : Nat.isZero(x))$ Caso borde, ya sabemos por consigna que se nos va a romper el $pred(zero)$ pero eso ocurre una vez que evaluamos. Por tanto, la expresión cumple con las condiciones de ser programa. Por que pred espera un Nat, y Zero lo es, y también vale que no hay variables libres. Concluimos que es programa y procedemos a su evaluación.
$(\lambda f : Nat \to Bool.f\ pred(zero))(\lambda x : Nat.isZero(x)) \xrightarrow{\beta} (\lambda x : Nat.isZero(x))\ pred(zero)) \xrightarrow{\beta} isZero(pred(zero)) \longrightarrow ERROR$
No tenemos en las reglas de semántica operacional definido $pred(zero)$ por tanto, esta evaluación concluye a un **estado de error, es decir, un estado de evaluación donde el término está en forma normal pero no es un valor**

8. $\mu y : Nat.succ(y)$ este último caso tiene una notación en cálculo $\lambda$ llamada especial donde $\mu$ es un *operador de punto fijo o recursión explícita*. Esto es, $\mu$ **devuelve el punto de fijo de una función**. La notación sería

$$\mu x.f(x) \equiv f(\mu x.f(x))$$

Con esto en mente, decimos que
$\mu y : Nat.succ(y) \longrightarrow succ(\mu y : Nat.succ(y)) \longrightarrow succ(succ(\mu y : Nat.succ(y))) \longrightarrow ...$
Como en este caso diverge ya que no termina nunca, no podemos evaluarlo.

# 4 Extensiones

## 4.1 Ejercicio 20 (pares o productos)

Este ejercicio extiende el cálculo-$\lambda$ tipado con pares. Las gramáticas de los tipos y los términos se extienden de la siguiente manera:

$$\tau ::= \cdots \mid \tau \times \tau$$
$$M ::= \cdots \mid \langle M, M \rangle \mid \pi_1(M) \mid \pi_2(M)$$

donde $\sigma \times \tau$ es el tipo de los pares cuya primera componente es de tipo $\sigma$ y cuya segunda componente es de tipo $\tau$, $\langle M, N \rangle$ construye un par y $\pi_1(M)$ y $\pi_2(M)$ proyectan la primera y la segunda componente de un par, respectivamente.
Nos piden definir las reglas de tipado para los nuevos constructores de términos. Con esas reglas y ciertos tipos, exhibir habitantes de las siguientes expresiones :

a) Definimos las reglas de tipado

$$\frac{\Gamma \vdash M_1 : \tau_1 \qquad \Gamma \vdash M_2 : \tau_2}{\Gamma \vdash \langle M_1, M_2 \rangle : \tau_1 \times \tau_2}\ \text{T-PAR}$$

$$\frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash \pi_1(M) : \tau_1}\ \text{T-FIRST}$$

$$\frac{\Gamma \vdash M : \tau_1 \times \tau_2}{\Gamma \vdash \pi_2(M) : \tau_2}\ \text{T-SECOND}$$

b) Veamos entonces las siguientes expresiones. Antes, repasemos lo que era un **término habitante del tipo**. Esto es, en simple palabras, **una lambda que tiene ese tipo**. La idea es proponer uno que cumpla y hacer la derivación para probarlo como hicimos en el ejercicio 9.

1. Constructor de pares : $\sigma \to \tau \to (\sigma \times \tau)$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x : \sigma, y : \tau \vdash x : \sigma}\ \text{ax} \qquad \dfrac{}{\Gamma, x : \sigma, y : \tau \vdash y : \tau}\ \text{ax}}{\Gamma, x : \sigma, y : \tau \vdash \langle x, y \rangle : \sigma \times \tau}\ \text{T-PAR}}{\Gamma, x : \sigma \vdash \lambda y : \tau.\langle x, y \rangle : \tau \to (\sigma \times \tau)}\ \text{T-ABS}}{\Gamma \vdash \lambda x : \sigma.\lambda y : \tau.\langle x, y \rangle : \sigma \to \tau \to (\sigma \times \tau)}\ \text{T-ABS}$$

2. Proyecciones : $(\sigma \times \tau) \to \sigma$ y $(\sigma \times \tau) \to \tau$
   - $(\sigma \times \tau) \to \sigma$

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : (\sigma \times \tau) \vdash x : (\sigma \times \tau)}\text{ ax}}{\Gamma, x : (\sigma \times \tau) \vdash \pi_1(x) : \sigma}\text{ T-FIRST}}{\Gamma \vdash \lambda x : (\sigma \times \tau).\pi_1(x) : (\sigma \times \tau) \to \sigma}\text{ T-ABS}$$

   - $(\sigma \times \tau) \to \tau$

$$\dfrac{\dfrac{\dfrac{}{\Gamma, x : (\sigma \times \tau) \vdash x : (\sigma \times \tau)}\text{ ax}}{\Gamma, x : (\sigma \times \tau) \vdash \pi_2(x) : \tau}\text{ T-SECOND}}{\Gamma \vdash \lambda x : (\sigma \times \tau).\pi_2(x) : (\sigma \times \tau) \to \tau}\text{ T-ABS}$$

3. Conmutatividad $(\sigma \times \tau) \to (\tau \times \sigma)$

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma, x : (\sigma \times \tau) \vdash x : (\sigma \times \tau)}\text{ ax}}{\Gamma, x : (\sigma \times \tau) \vdash \pi_2(x) : \tau}\text{ T-SECOND} \quad \dfrac{\dfrac{}{\Gamma, x : (\sigma \times \tau) \vdash x : (\sigma \times \tau)}\text{ ax}}{\Gamma, x : (\sigma \times \tau) \vdash \pi_1(x) : \sigma}\text{ T-FIRST}}{\Gamma, x : (\sigma \times \tau) \vdash \langle \pi_2(x), \pi_1(x) \rangle : (\tau \times \sigma)}\text{ T-PAR}}{\Gamma \vdash \lambda x : (\sigma \times \tau).\langle \pi_2(x), \pi_1(x) \rangle : (\sigma \times \tau) \to (\tau \times \sigma)}\text{ T-ABS}$$

4. Asociatividad $((\sigma \times \tau) \times \rho) \to (\sigma \times (\tau \times \rho))$ y $(\sigma \times (\tau \times \rho)) \to ((\sigma \times \tau) \times \rho)$
   - $((\sigma \times \tau) \times \rho) \to (\sigma \times (\tau \times \rho))$
     La idea es encontrar una lamda que tome algo como $\langle \langle x, y \rangle, z \rangle$ y evalue a $\langle x, \langle y, z \rangle \rangle$. Con esto en mente (lo escribo en color para que se vea mejor):

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma' \vdash x : (\sigma \times \tau) \times \rho}\text{ ax}}{\Gamma' \vdash \pi_1(x) : \sigma \times \tau}\text{ T-FIRST}}{\Gamma' \vdash \pi_1(\pi_1(x)) : \sigma}\text{ T-FIRST} \quad \dfrac{\dfrac{\dfrac{}{\Gamma' \vdash x : (\sigma \times \tau) \times \rho}\text{ ax}}{\Gamma' \vdash \pi_2(x) : \tau \times \sigma}\text{ T-SECOND y Conmutatividad}}{\Gamma' \vdash \pi_1(\pi_2(x)) : \tau}\text{ T-FIRST} \quad \dfrac{\dfrac{}{\Gamma' \vdash x : (\sigma \times \tau) \times \rho}\text{ ax}}{\Gamma' \vdash \pi_2(x) : \rho}\text{ T-SECOND}}{\Gamma' \vdash \langle \pi_1(\pi_2(x)), \pi_2(x) \rangle : (\tau \times \rho)}\text{ T-PAR}}{\Gamma, x : ((\sigma \times \tau) \times \rho) \vdash \langle \pi_1(\pi_1(x)), \langle \pi_1(\pi_2(x)), \pi_2(x) \rangle \rangle : (\sigma \times (\tau \times \rho))}\text{ T-PAR}}{\Gamma \vdash \lambda x : ((\sigma \times \tau) \times \rho).\ \langle \pi_1(\pi_1(x)), \langle \pi_1(\pi_2(x)), \pi_2(x) \rangle \rangle : ((\sigma \times \tau) \times \rho) \to (\sigma \times (\tau \times \rho))}\text{ T-ABS}$$

   - $(\sigma \times (\tau \times \rho)) \to ((\sigma \times \tau) \times \rho)$
     Muy parecido al item anterior
5. Currificación $((\sigma \times \tau) \to \rho) \to (\sigma \to \tau \to \rho)$ y $(\sigma \to \tau \to \rho) \to ((\sigma \times \tau) \to \rho)$

c) Para extender el conjunto de valores, agregamos el par que es un valor, donde cada una de sus componentes lo es :

$$V ::= \ldots \mid \langle V_1, V_2 \rangle$$

d) Las reglas de semántica operacional son las reglas que definen cómo los términos del programa se evaluan hasta llegar a un resultado, la estructura básica es :

$$\dfrac{M \to M'}{f(M) \to f(M')}\text{ E- nombre}$$

La idea es que si un término $M$ evalúa a $M'$, entonces la evaluación cumple que cierta función sobre el término evaluará a la misma función aplicada a $M'$. Por supuesto, hay casos que es directo porque $M$ es una forma normal. Veamos :

$$\dfrac{}{\pi_1(\langle V_1, V_2 \rangle) \to V_1}\text{ E-FIRSTVALUE}$$

$$\dfrac{}{\pi_2(\langle V_1, V_2 \rangle) \to V_2}\text{ E-SECONDVALUE}$$

Qué pasa si el término no es un valor y puede ser evaluado ... ?

$$\dfrac{M \to M'}{\pi_1(M) \to \pi_1(M')}\text{ E-FIRST}$$

$$\dfrac{M \to M'}{\pi_2(M) \to \pi_2(M')}\text{ E-SECOND}$$

Vamos con el constructor de pares

$$\frac{M \to M'}{\langle M, N\rangle \to \langle M', N\rangle}\ \text{E-PARES1}$$

$$\frac{N \to N'}{\langle M, N\rangle \to \langle M, N'\rangle}\ \text{E-PARES2}$$

Por qué no existe algo como ...

$$\frac{N \to N', M \to M'}{\langle M, N\rangle \to \langle M', N'\rangle}\ \text{E-PARES3}$$

La respuesta es que esta forma de evaluar rompería el determinismo, es decir, cada término tiene a lo sumo una regla que puede aplicarse en cada paso. Si hiciésemos algo de ese estilo estaríamos permitiendo que la reducción de términos tengan dos reglas diferentes que podrían aplicarse al mismo tiempo y produzcan resultados diferentes. Esto básicamente rompe el determinismo.

e) La propiedad de preservación de tipos afirma que si $\Gamma \vdash M : \tau$ y $M \to N$, entonces $\Gamma \vdash N : \tau$. Esto vale, ya que en nuestro caso todas las reglas de evaluación que postulamos, preservan el tipo de los términos.

La propiedad de progreso dice que si $\Gamma \vdash M : \tau$, entonces $M$ es un valor o existe un $N$ tal que $M \to N$. En nuestro caso, cualquier término bien tipado es un valor o puede reducirse

## 4.2 Ejercicio 22

Este ejercicio extiende el Cálculo Lambda tipado con listas. Comenzamos ampliando el conjunto de tipos:

$$\tau ::= \ldots \mid [\tau]$$

donde $[\tau]$ representa el tipo de las listas cuyas componentes son de tipo $\tau$. El conjunto de términos ahora incluye:

$$M, N, O ::= \ldots \mid []_\tau \mid M :: N \mid \text{case } M \text{ of } \{[] \rightsquigarrow N \mid h :: t \rightsquigarrow O\} \mid \textbf{foldr } M \text{ base} \rightsquigarrow N; \text{rec}(h, r) \rightsquigarrow O$$

donde:

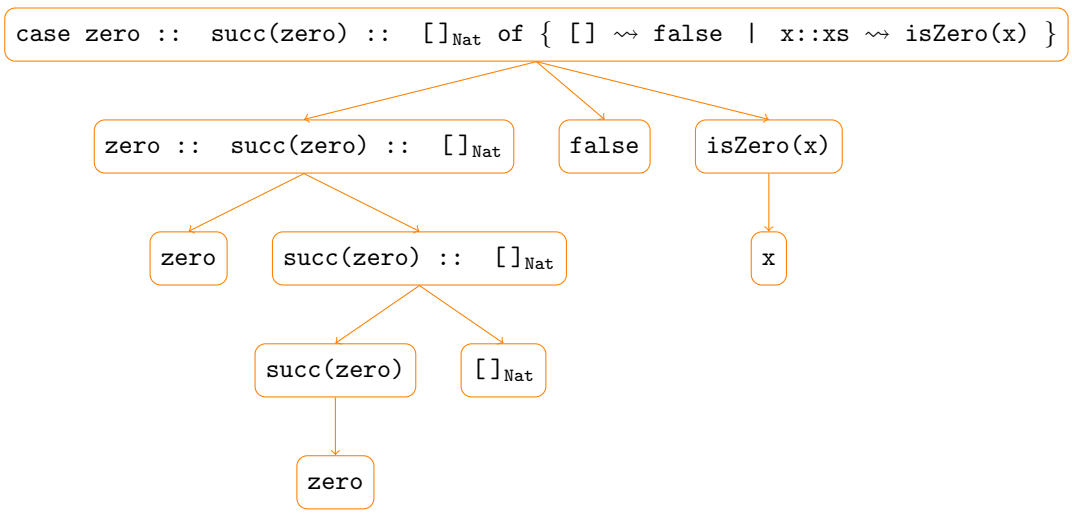
- $[]_\sigma$ es la lista vacía cuyos elementos son de tipo $\sigma$.
- $M :: N$ agrega $M$ al principio de la lista $N$.
- case $M$ of $\{[] \rightsquigarrow N \mid h :: t \rightsquigarrow O\}$ es el observador de listas. Por su parte, los nombres de variables que se indiquen luego del | ($h$ y $t$ en este caso) son variables que pueden aparecer libres en $O$ y se ligan con la cabeza y cola de la lista respectivamente.
- foldr $M$ base $\rightsquigarrow N$; rec$(h, r) \rightsquigarrow O$ es el operador de recursión estructural (no currificado). Los nombres de variables entre paréntesis ($h$ y $r$ en este caso) son variables que pueden aparecer libres en $O$ y deberán ser ligadas con la cabeza y el resultado de la recursión, respectivamente.

Por ejemplo:

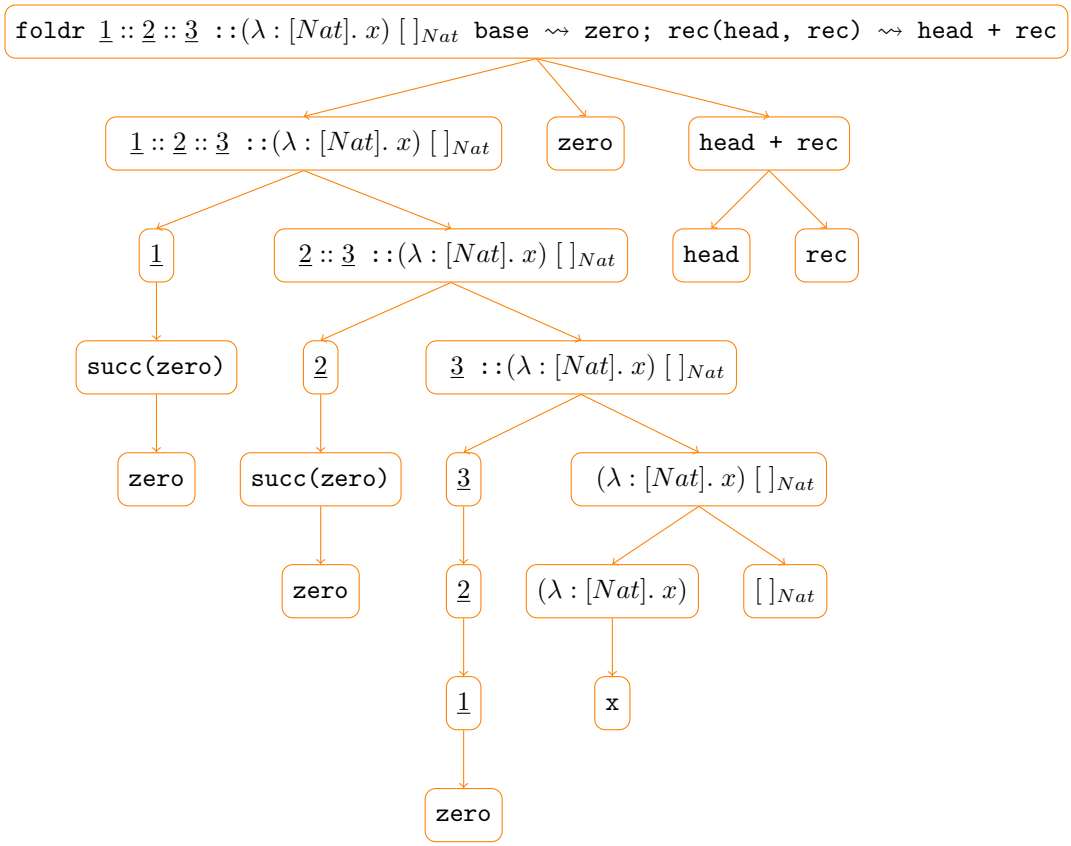
$$\text{case zero} :: \text{succ(zero)} :: []_{\text{Nat}} \text{ of } \{[] \rightsquigarrow \text{false} \mid x :: xs \rightsquigarrow \text{isZero}(x)\} \twoheadrightarrow \text{true}$$
$$\text{foldr } \underline{1} :: \underline{2} :: \underline{3} :: (\lambda x : [\text{Nat}].x)\ []_{\text{Nat}} \text{ base} \rightsquigarrow \text{zero};\ \text{rec}(\text{head}, \text{rec}) \rightsquigarrow \text{head} + \text{rec} \twoheadrightarrow \underline{6}$$

a) Mostrar el árbol sintáctico para los ejemplos

I. Case ... of ...



II. foldr




b) Agregar las reglas de tipado

$$\frac{}{\Gamma \vdash [\,]_\sigma : [\sigma]}\ \text{T-EMPTY}$$

$$\frac{\Gamma \vdash M : \sigma \qquad \Gamma \vdash N : [\sigma]}{\Gamma \vdash M :: N : [\sigma]}\ \text{T-APPEND}$$

$$\frac{\Gamma \vdash M : \sigma \qquad \Gamma, [\,] \vdash N : \sigma \qquad \Gamma, h : \sigma, t : [\sigma] \vdash O : \sigma}{\Gamma \vdash \text{case } M \text{ of } \{[\,] \rightsquigarrow N \mid h :: t \rightsquigarrow O\} : \sigma}\ \text{T-CASE}$$

$$\frac{\Gamma \vdash M : [\sigma] \qquad \Gamma \vdash N : \sigma \qquad \Gamma, h : \rho, r : \sigma \vdash O : \sigma}{\Gamma \vdash \text{foldr } M \text{ base} \rightsquigarrow N;\ \text{rec}(h, r) \rightsquigarrow O : \sigma}\ \text{T-FOLDR}$$

c) Demostrar el juicio de tipado

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash x : Bool}\,\text{ax} \quad \dfrac{\dfrac{}{\Gamma \vdash x : Bool}\,\text{ax} \quad \dfrac{}{\Gamma \vdash y : [Bool]}\,\text{ax}}{\Gamma \vdash x :: y : [Bool]}\,\text{T-APPEND}}{\Gamma \vdash x :: x :: y : [Bool]}\,\text{T-APPEND} \quad \dfrac{}{\Gamma \vdash y : [Bool]}\,\text{ax} \quad \dfrac{\dfrac{}{\Gamma' \vdash b : Bool}\,\text{ax} \quad \dfrac{}{\Gamma' \vdash a : [Bool]}\,\text{ax} \quad \dfrac{}{\Gamma' \vdash [\,]_{Bool} : [Bool]}\,\text{T-EMPTY}}{\Gamma, b : Bool, a : [Bool] \vdash \text{if } b \text{ then } a \text{ else } [\,]_{\text{Bool}} : [Bool]}\,\text{T-IF}}{x : \text{Bool},\ y : [\text{Bool}] \vdash \text{foldr } x :: x :: y \text{ base} \rightsquigarrow y;\ \text{rec}(b, a) \rightsquigarrow \text{if } b \text{ then } a \text{ else } [\,]_{\text{Bool}} : [\text{Bool}]}\,\text{T-FOLDR}$$

d) Extensión del conjunto de valores

$$V ::= \ldots \mid V :: V \mid [\,]_\sigma$$

e) Agregar las reglas de deducción

$$\frac{}{V :: N \to V :: N}\ \text{E-APPEND1}$$

$$\frac{M \to M'}{M :: N \to M' :: N}\ \text{E-APPEND2}$$

Veamos qué sucede con *case*. Esta función reducirá, dependiendo quién sea M, a una expresión o a otra, veamos primero los casos simples

$$\frac{}{case\ [\,]_\sigma\ of\{[\,] \rightsquigarrow N \mid h :: t \rightsquigarrow O\} \to N}\ \text{T-CASEEMPTY}$$

$$\frac{}{case\ V_1 :: V_2\ of\{[\,] \rightsquigarrow N \mid h :: t \rightsquigarrow O\} \to O\{h := V_1\}\{t := V_2\}}\ \text{T-CASEVALUES}$$

Si $M$ no es valor y se puede evaluar, $M \to M'$:

$$\frac{M \to M'}{case\ M\ of\{[\,] \rightsquigarrow N \mid h :: t \rightsquigarrow O\} \to case\ M'\ of\{[\,] \rightsquigarrow N \mid h :: t \rightsquigarrow O\}}\ \text{T-CASE}$$

Veamos *foldr*, primero los casos simples, cuando la lista a foldear es vacía o tiene valores

$$\frac{}{\text{foldr } [\,] \text{ base} \rightsquigarrow N;\ \text{rec}(h, r) \rightsquigarrow O \to N}\ \text{E-FOLDREMPTY}$$

$$\frac{}{\text{foldr } V_1 :: V_2 \text{ base} \rightsquigarrow N;\ \text{rec}(h, r) \rightsquigarrow O \to O\{h := V_1\}\{r := foldr\ V_2 \text{ base} \rightsquigarrow N;\ \text{rec}(h, r) \rightsquigarrow O\}}\ \text{E-FOLDRVALUES}$$

$$\frac{M \to M'}{\text{foldr } M \text{ base} \rightsquigarrow N;\ \text{rec}(h, r) \rightsquigarrow O \to \text{foldr } M' \text{ base} \rightsquigarrow N;\ \text{rec}(h, r) \rightsquigarrow O}\ \text{E-FOLDR}$$

### 4.3 Ejercicio 23

A partir de la extensión del ejercicio 22, definir una nueva extensión que incorpore expresiones de la forma `map(M, N)`, donde $N$ es una lista y $M$ una función que se aplicará a cada uno de los elementos de $N$.
Importante: tener en cuenta las anotaciones de tipos al definir las reglas de tipado y semántica.

La heurística es similar a lo que venimos haciendo. Primero, extendemos a los términos el nuevo, luego, extendemos las reglas de tipo, si falta agregar un valor, lo agregamos, finalmente, postulamos las reglas de la semántica operacional para saber cómo evaluar y $c'est\ fini$!.

- Agregamos el término
$$M, N ::= \ldots \mid map(M, N)$$
- Extendemos las reglas de tipo
$$\frac{\Gamma \vdash M : \tau \to \sigma \qquad \Gamma \vdash N : [\tau]}{\Gamma \vdash map(M, N) : [\sigma]}\ \text{T-MAP}$$
- No falta ningún valor
- Extendemos las reglas de semántica operacional
Qué pasa si $N$ es una lista vacía ? Recordar que N es una lista de tipo $\tau$ y esperamos evaluar a una lista de tipo $\sigma$. Es lo mismo si $M$ todavía no es valor ? Para la siguiente regla $M$ es un valor
$$\frac{}{map(M, [\,]_\tau) \to [\,]_\sigma}\ \text{E-MAPEMPTY}$$
Qué pasa si $M$ no es un valor ? En qué contexto podría pasar ? Una función que deba ser evaluada para devolver otra que sea la que se aplique al valor $\tau$ de $N$.
$$\frac{M \to M'}{map(M, N) \to map(M', N)}\ \text{E-MAP1}$$
Veamos si $M$ es un valor y la lista $N$ ya es un valor...
$$\frac{}{map(M, V_1 :: V_2) \to M\ V_1 :: map(M, V_2)}\ \text{E-MAPVALUES}$$
Último caso, qué pasa si $N$ se puede reducir ? En el contexto normal, donde la lista todavía no exhibe los valores como $V_1 :: V_2$ o $[\,]_\tau$
$$\frac{N \to N'}{map(M, N) \to map(M, N')}\ \text{E-MAP2}$$