Paradigmas (de Lenguajes) de Programación

Apunte de las clases de cálculo lambda

Alejandro Díaz-Caro (Jano)

Versión: 2024-05-18

Aclaración importante

Estos apuntes de clase, son los apuntes que armé para dar la clase. No pretenden reemplazar al libro. Simplemente los apuntes están armados en PDF porque es más prolijo para seguirlos, y ya que los tengo, se los comparto por si les sirve. No reemplazan ni al libro, ni a la clase. Siempre es mejor ir a la clase y leer el libro. Y si algo no se entiende en el apunte, vayan al libro y si tampoco se entiende en el libro, consulten, que para eso estamos. Jano

Segunda aclaración (menos importante quizá)

No, no hay slides de estas clases. Prefiero el "pizarrón" (así sea un pizarrón electrónico), porque me obliga a ir más lento y a explicar mejor que si ya está todo escrito en las slides... más cuando hay tantos simbolitos nuevos y desconocidos por ustedes. Es intencional. Estas clases son sin slides. Jano

Bibliografía sugerida

- G. Dowek y J.-J. Lévy, "Introduction to the theory of programming languages", Springer, 2011.
- D. P. Friedman, M. Wand y C. T. Hayner, "Essentials of programming languages", MIT Press, 2011.
- G. Winskel, "Lecture Notes on Denotational Semantics", Cambridge, 2005.

Índice

1. Clase 1 **2**

1.1. Cálculo lambda con booleanos . . . 2

1.1.1. Primeras definiciones . . . 2

1.1.2. Gramática del cálculo lambda con booleanos . . . 3

1.1.3. Semántica operacional . . . 3

1.1.4. Captura de variables . . . 4

1.2. Estrategias de reducción . . . 5

1.2.1. Primeras definiciones . . . 5

1.2.2. Reducción débil . . . 7

1.2.3. Call-by-name (CBN) . . . 7

1.2.4. Call-by-value (CBV) . . . 8

1.3. Tipos simples . . . 8

1.3.1. Introducción 8
1.3.2. Gramática de cálculo lambda con booleanos tipado 9
1.3.3. La relación de tipado 10

2. Clase 2 **11**
2.1. Polimorfismo 11
2.1.1. Introducción 11
2.1.2. Sistema F 13
2.1.3. Polimorfismo let 13
2.2. Recursión 16
2.2.1. No terminación 17
2.2.2. Tipado del punto fijo 18

3. Clase 3 **18**
3.1. Interpretación 18
3.1.1. Interpretación en CBN 19
3.1.2. Interpretación en CBV 20
3.2. Semántica denotacional 20
3.2.1. Primeras definiciones 20
3.2.2. La semántica denotacional de cálculo lambda tipado 22

1. Clase 1

1.1. Cálculo lambda con booleanos

1.1.1. Primeras definiciones

- Variables (usamos letras minúsculas del final del abecedario x, y, z).
- Dos construcciones de base:
  - Construcción explícita de función (sin nombre): $\lambda x.M$.
  - Aplicación de una función a un argumento: MN.
- Una constante para cada valor de verdad: true y false.

> En este apunte usaremos **tt** y **ff**, sólo por cuestión de espacio.
> En las prácticas, usaremos true y false.

- Test: if M then N else O.

Observaciones.

- Todo es función: Una función tomando un argumento constante u otra función, es lo mismo. Por ejemplo, la función que toma una función y la compone consigo misma, es

$$\lambda f.\lambda x.f(fx)$$

- No existen funciones que tomen varios argumentos. Por ejemplo,

$$f(x, y) = \text{if } y \text{ then } x \text{ else } \mathbf{tt}$$

en cálculo lambda se escribe

$$\underbrace{\lambda y.\lambda x.\text{if } x \text{ then } y \text{ else } \mathbf{tt}}_{F}$$

$F\mathbf{ff}$ es una función que espera un argumento para dar su negación y $F\mathbf{tt}$ es una función que espera un argumento y devuelve siempre $\mathbf{tt}$.

- La aplicación asocia a la izquierda: $F\mathbf{ff}\mathbf{tt} = (F\mathbf{ff})\mathbf{tt}$.

1.1.2. Gramática del cálculo lambda con booleanos

Definición 1.1 (Gramática del cálculo lambda con booleanos) Los términos válidos de cálculo lambda con booleanos los podemos describir con una gramática formal:

$$M ::= x \mid \lambda x.M \mid MM \mid \mathbf{tt} \mid \mathbf{ff} \mid \text{if } M \text{ then } M \text{ else } M$$

El cálculo lambda (sin necesidad de los booleanos) es Turing completo, es decir que todas las funciones de enteros computables son programables en él. A los booleanos los agregamos sólo por practicidad (sino deberíamos estar codificándolos en el lenguaje)

1.1.3. Semántica operacional

La "gramática" nos dice qué términos podemos escribir sintácticamente. La "semántica" (operacional en este caso) nos da el significado de los términos, al definir como operan. Las siguientes reglas tienen la forma $M \to N$, y se lee "M reduce a N" o "M se reescribe como N".

Definición 1.2 (Semántica operacional)

$$(\lambda x.M)N \to M\{x := N\} \qquad (\beta \text{ reducción})$$
$$\text{if } \mathbf{tt} \text{ then } M \text{ else } N \to M \qquad (\text{if}_\text{t})$$
$$\text{if } \mathbf{ff} \text{ then } M \text{ else } N \to N \qquad (\text{if}_\text{f})$$

También tendremos reglas de congruencia que permitirán reducir un término dentro de otro.
Si $M \to N$, entonces

$$MO \to NO \qquad (\text{app}_l)$$
$$OM \to ON \qquad (\text{app}_r)$$
$$\lambda x.M \to \lambda x.N \qquad (\text{fun})$$
$$\text{if } M \text{ then } O \text{ else } P \to \text{if } N \text{ then } O \text{ else } P \qquad (\text{if}_c)$$

Ejercicio 1.3. Para pensar: Faltarían dos reglas... ¿cuáles son? ¿tienen sentido?

Observación. La regla (fun), que permite reducir dentro de la función, corresponde a la posibilidad de optimizar programas.

1.1.4. Captura de variables

Ejercicio 1.4.

1. $(\lambda x.\lambda x.x)\mathbf{tt}\mathbf{ff}$
2. $(\lambda x.\lambda y.(\lambda x.\mathsf{if}\ x\ \mathsf{then}\ x\ \mathsf{else}\ y)x)\mathbf{tt}\mathbf{ff}$
3. Imaginemos que en lugar de $(\lambda x.M)N$ escribimos $\mathsf{let}\ x = N\ \mathsf{in}\ M$. Como una notación particular para lo mismo.

   Ahora pensemos este caso:

   $$\left.\begin{array}{l}\mathsf{let}\ x = \mathbf{ff}\ \mathsf{in}\\ \quad \mathsf{let}\ f = \lambda y.\mathsf{if}\ y\ \mathsf{then}\ x\ \mathsf{else}\ \mathbf{ff}\ \mathsf{in}\\ \quad\quad \mathsf{let}\ x = \mathbf{tt}\ \mathsf{in}\\ \quad\quad\quad f\mathbf{tt}\end{array}\right\}\ \text{En las primeras versiones de Lisp, este ejemplo daba } \mathbf{tt} \text{ en lugar de } \mathbf{ff}.$$

Tenemos que definir precisamente qué significa $M\{x := N\}$. Lo definimos por inducción en M.

$$\begin{array}{rll} x\{x := N\} = N & & \\ y\{x := N\} = y & & \\ (\lambda x.M)\{x := N\} = \lambda x.M & & \\ (\lambda y.M)\{x := N\} = \lambda y.M\{x := N\} & & \text{Si } y \notin \mathsf{FV}(N) \\ (\lambda y.M)\{x := N\} = \lambda z.M\{y := z\}\{x := N\} & & \text{Si } y \in \mathsf{FV}(N) \\ (MO)\{x := N\} = M\{x := N\}O\{x := N\} & & \\ \mathbf{tt}\{x := N\} = \mathbf{tt} & & \\ \mathbf{ff}\{x := N\} = \mathbf{ff} & & \end{array}$$

$$(\mathsf{if}\ M\ \mathsf{then}\ O_1\ \mathsf{else}\ O_2)\{x := N\} = \mathsf{if}\ M\{x := N\}\ \mathsf{then}\ O_1\{x := N\}\ \mathsf{else}\ O_2\{x := N\}$$

Ejercicio 1.5. Definir, por inducción sobre M, $\mathsf{FV}(M)$.

1.2. Estrategias de reducción

1.2.1. Primeras definiciones

Definición 1.6 Notamos $\to^*$ al cierre reflexivo y transitivo de $\to$. Es decir, si $M \to^* N$, entonces, $M = O_0 \to O_1 \to O_2 \to \cdots \to O_n = N$, con $n \geq 0$. Notamos $\to^+$ al cierre transitivo de $\to$. Es decir, si $M \to^* N$, $M = O_0 \to O_1 \to O_2 \to \cdots \to O_n = N$, con $n \geq 1$.

Ejemplo 1.7

$$(\lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt})\mathbf{tt} \to^* \mathbf{ff}$$

porque

$$(\lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt})\mathbf{tt} \to \mathsf{if}\ \mathbf{tt}\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt} \to \mathbf{ff}$$

También,

$$(\lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt})\mathbf{tt} \to^+ \mathbf{ff}$$

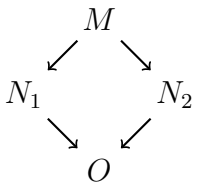
ya que $(\lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt})\mathbf{tt} \neq \mathbf{ff}$.

Definición 1.8

1. Un término M está en forma normal si no existe N tal que $M \to N$.
2. Un término M es normalizable (o tiene forma normal) si existe N en forma normal tal que $M \to^* N$.
3. Un término M es fuertemente normalizable si no existe una secuencia infinita N_0, N_1, ... tal que $M \to N_0 \to N_1 \to \ldots$. Es decir, toda secuencia de reducción comenzada en M debe ser finita y terminar en un término en forma normal.

Definición 1.9 Sea $\to_R$ una relación binaria, y $\to_R^*$ su cierre reflexivo y transitivo.

- $\to_R$ satisface la propiedad del diamante si $M \to_R N_1$ y $M \to_R N_2$ implica que existe O tal que $N_1 \to_R O$ y $N_2 \to_R O$.



- $\to_R$ es Church-Rosser o confluente si $\to_R^*$ satisface la propiedad del diamante. Es decir, si $M \to_R^* N_1$ y $M \to_R^* N_2$ implica que existe $=$ tal que $N_1 \to_R^* O$ y $N_2 \to_R^* O$.
- $\to_R$ tiene formas normales únicas si $M \to_R^* N_1$ y $M \to_R^* N_2$, para términos en forma normal N_1 y N_2, implica $N_1 = N_2$.

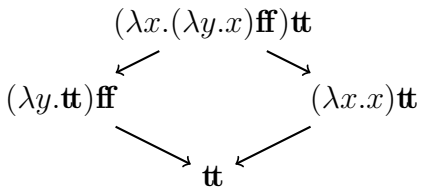
Lema 1.10

1. Si $\to_R$ satisface la propiedad del diamante, entonces es Church-Rosser.
2. Si $\to_R$ es Church-Rosser, entonces tiene formas normales únicas.

Demostración.
Ejercicio 1.11. Pensar cómo probar este lema. □

Teorema 1.12 La relación definida en la Sección 1.1.3 (semántica operacional de cálculo lambda con booleanos) es Church-Rosser. □

Ejemplo 1.13



Pero esta propiedad, cuando hay términos que no terminan, no es suficiente.

Ejemplo 1.14

$$\Omega = (\lambda x.xx)(\lambda y.yy)$$

Entonces

$$\Omega \to \Omega \to \Omega \to \cdots$$

¡No termina nunca!
Por otro lado, $(\lambda x.\mathbf{tt})\Omega$ reduce a sí mismo, y también reduce a $\mathbf{tt}$.
$(\lambda x.\mathbf{tt})\Omega$ tiene un único resultado, pero no cualquier camino llega a él.
Solución (en este caso): Si hay un lambda aplicado, reducir primero esa aplicación y no el argumento. Ésto, como veremos luego, es una estrategia.

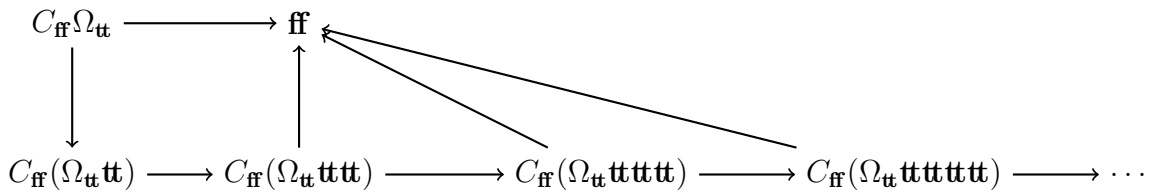
Ejemplo 1.15

$$\Omega_{\mathbf{tt}} = (\lambda x.xx\mathbf{tt})(\lambda y.yy\mathbf{tt})$$
$$C_{\mathbf{ff}} = (\lambda x.\mathbf{ff})$$

Entonces,

$$\Omega_{\mathbf{tt}} \to \Omega_{\mathbf{tt}}\mathbf{tt} \to \Omega_{\mathbf{tt}}\mathbf{tttt} \to \ldots$$

Por lo tanto, tenemos el siguiente diagrama



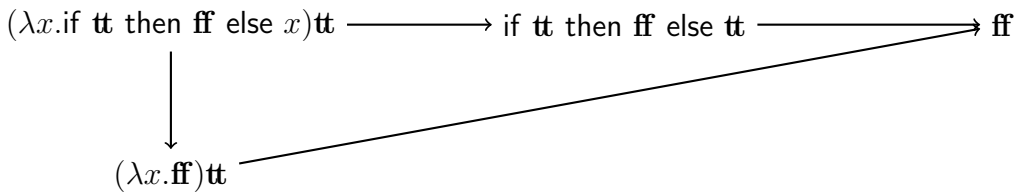
Este ejemplo, en algunos lenguajes termina, y en otros no termina nunca. Depende de la estrategia de reducción que usa el lenguaje.

La noción de estrategia de reducción permite definir el orden en el cual se debe reducir un término.

Definición 1.16 Llamamos redex a un subtérmino de un término que puede reducir (del inglés "reducible expression").

1.2.2. Reducción débil

Ejemplo motivador:



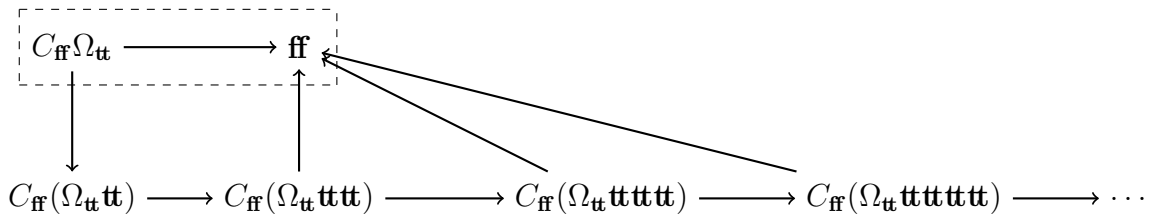
- La dirección $\to$ dice qué sucede cuando se ejecuta el programa.
- La dirección $\downarrow$ comienza a ejecutar el programa antes de recibir los argumentos, es decir, no ejecuta el programa sino que lo optimiza.

Definición 1.17 Una estrategia de reducción es débil si no reduce nunca el cuerpo de una función, es decir, si no reduce "bajo" λ.

Observación. La estrategia débil no optimiza programa, los ejecuta. Sólo hace falta para ésto eliminar la regla (fun):

$$\text{Si } M \to N \text{ entonces } \lambda x.M \to \lambda x.N$$

1.2.3. Call-by-name (CBN)



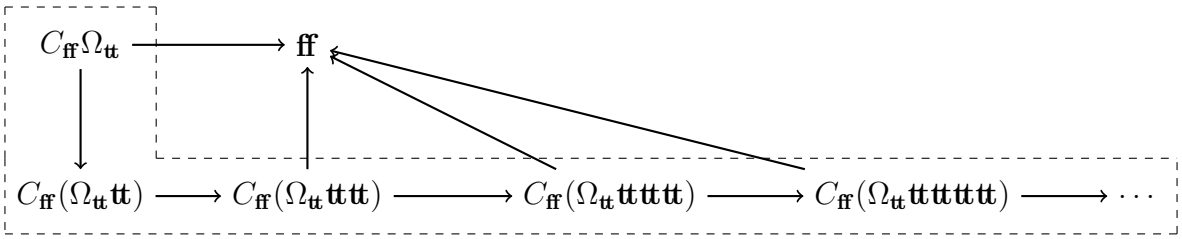
Definición 1.18 La estrategia call-by-name reduce siempre el redex más a la izquierda. En caso de ser además débil, será el más a la izquierda que no esté bajo un λ.

Teorema 1.19 (Estandarización) Si un término reduce a un término en forma normal, entonces la estrategia call-by-name termina. □

Una ventaja de ésta estrategia es el teorema de estandarización. Otra ventaja es que si tenemos, por ejemplo $\lambda x.\mathbf{ff}$ aplicado a un programa N que tiene un cálculo muy largo para llegar a su resultado, no necesitamos calcularlo. Por otro lado, si tenemos $(\lambda x.Mxx)N$, tendremos que calcular N dos veces.
De todas maneras, la mayoría de los lenguajes que usan call-by-name usan alguna manera de "compartir" información (por ejemplo, con punteros que dicen que $(\lambda x.Mxx)N$ reduce a Mxx, donde x es un puntero a N. A eso se le llama reducción lazy.

Ejercicio 1.20. Escribir las reglas de reducción y congruencia que implementan call-by-name.

1.2.4. Call-by-value (CBV)



Definición 1.21 A los términos M tales que $\mathsf{FV}(M) = \emptyset$ y que M esté en forma normal, se les llaman valores.

Definición 1.22 La estrategia call-by-value consiste en evaluar siempre los argumentos antes de pasarlos a la función. La idea es que

$$(\lambda x.M)V$$

reduce sólo cuando V esté en forma normal (si la estrategia es débil, y sólo reducimos términos cerrados, V es un valor).

En $(\lambda x.Mxx)N$ comenzamos por reducir N obteniendo un resultado, y recién ahí lo pasamos a la función. De esa manera el resultado de N es calculado una vez.

Ejercicio 1.23. Escribir las reglas de reducción y congruencia que implementan call-by-value.

Observación. Un poco de pereza es necesaria: if siempre debe evaluar primero la condición, estemos en call-by-name o call-by-value.

1.3. Tipos simples

1.3.1. Introducción

Ejemplos motivadores:

$$(\lambda x.\mathsf{if}\ x\ \mathsf{then}\ M\ \mathsf{else}\ N)\lambda y.y \to \mathsf{if}\ \lambda y.y\ \mathsf{then}\ M\ \mathsf{else}\ N \not\to$$
$$(\lambda x.x)\mathbf{tt}\lambda x.x \to \mathbf{tt}\lambda x.x \not\to$$

¡Todo es aplicable a todo! Sin restricciones. Hacer un if sobre una función no aplicada a nada no tiene sentido, ni tampoco pasarle un argumento a un booleano.
Idea: detectar este tipo de errores sintácticamente. Por ejemplo:

$$\frac{\lambda x.x \text{ recive un argumento y devuelve lo mismo} \quad \mathbf{tt} \text{ es un booleano}}{(\lambda x.x)\mathbf{tt} \text{ es un booleano}}$$

Es decir, deducimos que no tiene sentido pasarle un argumento a $(\lambda x.x)\mathbf{tt}$, ya que es un booleano, y lo dedujimos sin tener que ejecutar el programa.

En matemáticas:

$$\text{Función: Dominio} \longrightarrow \text{Codominio}$$
$$\text{Cualquier conjunto}$$

Ejemplo:

$$f : \mathit{Pares} \to \mathbb{N}$$
$$f(x) \mapsto \frac{x}{2}$$

¿Está bien definido $f(3+(4+1))$? Hay que determinar si $3+(4+1)$ pertenece al dominio, es decir, si es par.

En general, determinar si un objeto cualquiera pertenece a un conjunto cualquiera es un problema indecidible.

De todas maneras, $\frac{x}{2}$ lo podemos calcular si x es un número (y no, por ejemplo, una función). Así que vamos a restringir las clases de conjuntos que se pueden utilizar como dominios. A estos conjuntos los llamamos **tipos**.

1.3.2. Gramática de cálculo lambda con booleanos tipado

Definición 1.24 (Tipos simples) Los tipos de cálculo lambda con booleanos los definimos inductivamente por:

- `Bool` es un tipo.
- Si τ y σ son tipos, $\tau \to \sigma$ es un tipo que representa las funciones de τ en σ.

Teniendo tipos, tendremos que escribir de qué tipo son cada una de las variables. Como sólo nos vamos a interesar en términos sin variables libres (sólo los subtérminos tendrán variables libres), es suficiente con escribir el tipo cuando se liga la variable. Por ejemplo, en lugar de $\lambda x.x$ la función identidad es una para cada tipo:

- λx:`Bool`$.x$ es la identidad sobre los booleanos
- λx:`Bool` $\to$ `Bool`$.x$ es la identidad sobre las funciones de booleanos en booleanos.

La gramática de cálculo lambda con booleanos tipado la definimos con una gramática de tipos y una de términos, de la siguiente manera:

$$\tau ::= \texttt{Bool} \mid \tau \to \tau$$
$$M ::= x \mid \lambda x{:}\tau.M \mid MM \mid \mathbf{tt} \mid \mathbf{ff} \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M$$

1.3.3. La relación de tipado

Queremos definir inductivamente la relación $M : \tau$ (es decir, el término M tiene el tipo τ). Pero si hay variables libres en M, ¿cómo las tipo?[1].
Por ejemplo:

$$\lambda x{:}\mathtt{Bool}.yx$$

¿Qué tipo tiene y? Claramente tiene que ser una función de $\mathtt{Bool}$ en algo, pero ¿cómo defino ese algo?

Contextos Un contexto nos da tipos para variables, entonces, en vez de decir $\lambda x{:}\mathtt{Bool}.yx : \mathtt{Bool} \to \mathtt{Bool}$, decimos, si $y : \mathtt{Bool} \to \mathtt{Bool}$, entonces $\lambda x{:}\mathtt{Bool}.yx : \mathtt{Bool} \to \mathtt{Bool}$. La notación que usamos es la siguiente:

$$\underbrace{y{:}\mathtt{Bool} \to \mathtt{Bool}}_{\text{contexto}} \vdash \lambda x{:}\mathtt{Bool}.yx : \mathtt{Bool} \to \mathtt{Bool}$$

Genéricamente, queremos definir la relación $\Gamma \vdash M : \tau$ que asocia un término M y un contexto Γ a un tipo τ.

Definición 1.25 (Sistema de tipos) La relación de tipado $\Gamma \vdash M : \tau$ se define inductivamente por:

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ ax_v \qquad \frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x{:}\tau.M : \tau \to \sigma} \to_i \qquad \frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma} \to_e$$

$$\frac{}{\Gamma \vdash \mathbf{tt} : \mathtt{Bool}}\ ax_t \qquad \frac{}{\Gamma \vdash \mathbf{ff} : \mathtt{Bool}}\ ax_f \qquad \frac{\Gamma \vdash M : \mathtt{Bool} \quad \Gamma \vdash N_1 : \tau \quad \Gamma \vdash N_2 : \tau}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : \tau}\ \mathtt{if}$$

Ejemplo 1.26 Vamos a tipar el término $\lambda x{:}\mathtt{Bool}.\mathsf{if}\ x\ \mathsf{then}\ (\lambda y{:}\mathtt{Bool}.y)x\ \mathsf{else}\ \mathbf{ff}$.
Sea $\Gamma = x : \mathtt{Bool}$ y $\Delta = x : \mathtt{Bool}, y : \mathtt{Bool}$. Entonces,

$$\dfrac{\dfrac{}{\Gamma \vdash x : \mathtt{Bool}}\ ax_v \quad \dfrac{\dfrac{\dfrac{}{\Delta \vdash y : \mathtt{Bool}}\ ax_v}{\Gamma \vdash \lambda y{:}\mathtt{Bool}.y : \mathtt{Bool} \to \mathtt{Bool}} \to_i \quad \dfrac{}{\Gamma \vdash x : \mathtt{Bool}}\ ax_v}{\Gamma \vdash (\lambda y{:}\mathtt{Bool}.y)x : \mathtt{Bool}} \to_e \quad \dfrac{}{\Gamma \vdash \mathbf{ff} : \mathtt{Bool}}\ ax_f}{\dfrac{\Gamma \vdash \mathsf{if}\ x\ \mathsf{then}\ (\lambda y{:}\mathtt{Bool}.y)x\ \mathsf{else}\ \mathbf{ff} : \mathtt{Bool}}{\vdash \lambda x{:}\mathtt{Bool}.\mathsf{if}\ x\ \mathsf{then}\ (\lambda y{:}\mathtt{Bool}.y)x\ \mathsf{else}\ \mathbf{ff} : \mathtt{Bool} \to \mathtt{Bool}} \to_i}\ \mathtt{if}$$

Ejercicio 1.27. Tipar $(\lambda x.\mathsf{if}\ \mathbf{tt}\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ x)\mathbf{tt}$

[1] Dijimos que solo vamos a interesarnos por términos cerrados (sin variables libres), pero como buscamos una definición inductiva, necesitamos poder tipar cada subtérmino, y un subtérmino de un término sin variables libres, puede perfectamente tener variables libres.

Ejercicio 1.28. Tipar $\lambda x{:}\tau.xx$ para algún τ.

Teorema 1.29 (Subject reduction o preservación de tipos)
Si $\Gamma \vdash M : \tau$ y $M \to N$ entonces $\Gamma \vdash N : \tau$. □

Es decir: si deducimos el tipo τ para un término (usando la Definición 1.25), o sea, sin ejecutar el programa, y luego ejecutamos el programa obteniendo N, entonces el término N tiene el mismo tipo. ¡Es exactamente lo que queríamos! La intención fue desde el principio saber qué tipo de resultado voy a tener al ejecutar un programa (un número, una función, etc), y este teorema nos dice que el sistema de tipos que definimos hace eso.

Ejercicio 1.30. Demostrar el Teorema 1.29.
Ayuda. Primero demostrar el siguiente lema, por inducción en M:

Lema 1.31 (Sustitución) Si $\Gamma, x : \tau \vdash M : \sigma$ y $\Gamma \vdash N : \tau$, entonces $\Gamma \vdash M\{x := N\} : \sigma$.

Luego, usando ese lema, mostrar el teorema por inducción en la definición de $\to$ (Definición 1.2).

Teorema 1.32 (Teorema de Tait o normalización fuerte)
Todo término tipado en este sistema, termina. □

¿Qué sucede con $\Omega = (\lambda x{:}\tau.xx)(\lambda x{:}\sigma.xx)$? No es tipable. Es decir, no existen tipos τ, σ, y ρ tal que $\vdash \Omega : \rho$.

Ejercicio 1.33. Extender cálculo lambda con constructores para pares: (M, N), $\mathsf{fst}(M, N)$ y $\mathsf{snd}(M, N)$. Dar la gramática, semántica operacional y reglas de tipado.

2. Clase 2

2.1. Polimorfismo

2.1.1. Introducción

En la sección anterior vimos que a $\lambda x{:}\tau.x$ podemos derivarle el tipo $\tau \Rightarrow \tau$, cualquiera sea el tipo τ. Podemos entonces introducir una variable de tipo X, y atribuirle a $\lambda x{:}X.x$ el tipo

$$\forall X.X \Rightarrow X$$

agregando una regla de tipado a las reglas de la Definición 1.25 según la cual si M tiene tipo $\forall X.\tau$, entonces M tiene tipo $\tau\{X := \sigma\}$ para cualquier σ.
A lenguajes con estas características se los llama polimórficos.

Ejemplo 2.1 Supongamos que queremos aplicar el térmnino $M = \lambda x{:}\tau.xx$ a la función identidad $I = \lambda x{:}\sigma.x$, para algún τ y σ. Entonces

$$MI \to II \to I$$

Vemos que la identidad recibe a la identidad como parámetro. ¿Quienes serían entonces τ y σ?
Si I es $\lambda x{:}\sigma.x$, y tiene tipo $\sigma \Rightarrow \sigma$, para algún σ, entonces $II = (\lambda x{:}\sigma.x)(\lambda x{:}\sigma.x)$ no está bien tipado.
En cambio, si permitiéramos diferentes tipos, $I_1 = \lambda x{:}\sigma_2.x$ y $I_2 = \lambda x{:}\sigma_1.x$, entonces $I_1 I_2$ está bien tipado para alguna elección de σ_2 y σ_1.

Ejercicio 2.2.

1. Tratar de tipar $II = (\lambda x{:}\sigma.x)(\lambda x{:}\sigma.x)$ con el sistema de la Definición 1.25.
2. Tipar $I_1 I_2$ con el sistema de la Definición 1.25, determinando quiénes son σ_1 y σ_2.

La diferencia entre II y $I_1 I_2$ es que en el primero, I debe tener el mismo tipo en ambas instancias, en cambio en el segundo ejemplo, cada función identidad puede instanciar su tipo de manera diferente.
Con un tipo $\forall$, podemos hacer que el I de II tenga tipo $\forall X.X \Rightarrow X$ y cada ocurrencia de I se instancie en un tipo diferente, haciendo que el término II pueda ser tipado como $\forall X.X \Rightarrow X$.
Veámoslo un poco más sistemático. Agreguemos dos reglas de tipado a las de la Definición 1.25:

$$\frac{\Gamma \vdash M : \tau}{\Gamma \vdash \lambda x{:}\tau.M : \forall X.\tau}\ \forall_i \quad \leftarrow \text{(mal, ya veremos porqué)} \qquad \frac{\Gamma \vdash M : \forall X.\tau}{\Gamma \vdash M : \tau\{X := \sigma\}}\ \forall_e$$

Entonces podemos tipar II de la siguiente manera:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{x{:}X \vdash x : X}\ ax_v}{\vdash \lambda x{:}X.x : X \Rightarrow X}\ \Rightarrow_i}{\vdash \lambda x{:}X.x : \forall X.X \Rightarrow X}\ \forall_i}{\vdash \lambda x{:}X.x : (\forall X.X \Rightarrow X) \Rightarrow (\forall X.X \Rightarrow X)}\ \forall_e \quad \dfrac{\dfrac{\dfrac{}{x{:}X \vdash x : X}\ ax_v}{\vdash \lambda x{:}X.x : X \Rightarrow X}\ \Rightarrow_i}{\vdash \lambda x{:}X.x : \forall X.X \Rightarrow X}\ \forall_i}{\vdash (\lambda x.{:}Xx)(\lambda x.{:}X)x : \forall X.X \Rightarrow X}\ \Rightarrow_e$$

El ejemplo anterior no es un ejemplo aislado. Podríamos por ejemplo hacer una función `map` para árboles, sea cual sea el tipo de los elementos en los árboles, lo que permite reusabilidad de código y por lo tanto programas más concisos.
Ahora vamos a resolver el problema de la regla $\forall_i$. ¿Qué está mal en la regla propuesta para la introducción de $\forall$? El siguiente ejemplo muestra el problema.

Ejemplo 2.3 (Problema con un $\forall_i$ descuidado)

$$\dfrac{\dfrac{\dfrac{}{x{:}X \vdash x : X}\ ax_v}{x{:}X \vdash x : \forall X.X}\ \forall_i}{\vdash \lambda x{:}X.x : X \Rightarrow \forall X.X}\ \Rightarrow_i \quad \leftarrow \text{(Ojo, esta derivación es un ejemplo de algo que está mal)}$$

En este caso, x tiene tipo X, pero X no está ligado por el $\forall$. Es una identidad que recibe un argumento de tipo "variable X" y devuelve cualquier tipo. No tiene sentido.

Solución: La regla $\forall_i$ tiene que pedir que la variable que se está ligando con el $\forall$, no esté libre en el contexto. Sí, volvemos a la noción de libres y ligadas, pero ahora en el contexto. Por ejemplo, si el contexto contiene $x : \forall X.\tau$, puedo usar la X ya que está en el contexto, pero ligada, no libre. Si el contexto contiene $x : X$, no puedo, ya que está libre.

2.1.2. Sistema F

Los tipos en Sistema F son los mismos de tipos simples (ver Definición 1.24) , a los que se agregan variables y para-todo ligando esas variables:

$$\tau ::= X \mid \texttt{Bool} \mid \tau \Rightarrow \tau \mid \forall X.\tau$$

A continuación damos las reglas de Sistema F. Como se puede apreciar, las reglas son exactamente las mismas que las de tipos simples (ver Definición 1.25), a las que se agregan las reglas de introducción y eliminación del para-todo.

Definición 2.4 (Sistema F)

$$\frac{}{\Gamma, x : \tau \vdash x : \tau}\ ax_v \qquad \frac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x{:}\tau.M : \tau \to \sigma} \to_i \qquad \frac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma} \to_e$$

$$\frac{}{\Gamma \vdash \mathbf{tt} : \texttt{Bool}}\ ax_t \qquad \frac{}{\Gamma \vdash \mathbf{ff} : \texttt{Bool}}\ ax_f \qquad \frac{\Gamma \vdash M : \texttt{Bool} \quad \Gamma \vdash N_1 : \tau \quad \Gamma \vdash N_2 : \tau}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : \tau}\ \texttt{if}$$

$$\frac{\Gamma \vdash M : \tau \quad X \notin \mathsf{FV}(\Gamma)}{\Gamma \vdash M : \forall X.\tau}\ \forall_i \qquad \frac{\Gamma \vdash M : \forall X.\tau}{\Gamma \vdash M : \tau\{X := \sigma\}}\ \forall_e$$

Donde si $\Gamma = x_1 : \tau_1, \ldots, x_n : \tau_n$, entonces $\mathsf{FV}(\Gamma) = \mathsf{FV}(\tau_1) \cup \cdots \cup \mathsf{FV}(\tau_n)$.
¿Quién es $\mathsf{FV}(\tau)$?

Ejercicio 2.5. Definir $\mathsf{FV}(\tau)$. Tip: sólo $\forall$ liga variables.

2.1.3. Polimorfismo let

Con tipos simples, teníamos un sistema "dirigido por sintaxis", esto es, dado un término, sabíamos exactamente qué regla de tipado aplicar. Si es una variable, aplicamos ax_v, si es una abstracción, aplicamos $\Rightarrow_i$, si es una aplicación, aplicamos $\Rightarrow_e$, etc.
En Sistema F perdimos esta propiedad. Si tengo una variable, podría necesitar aplicar ax_v o $\forall_e$, si tengo una abstracción, podría necesitar aplicar $\Rightarrow_i$, $\forall_i$, o $\forall_e$, etc. Sistema F no es dirigido por sintaxis, por lo cual, inferir un tipo para un término no es trivial. De hecho, no existe un algoritmo de inferencia que lo pueda hacer. Es indecidible.
Por este motivo, vamos a dar otro sistema polimórfico, menos general, pero 1) dirigido por sintaxis y 2) decidible. Este sistema es el polimorfismo let. Primero, vamos a extender el cálculo lambda con un nuevo término, $\mathsf{let}\ x = N\ \mathsf{in}\ M$, que es equivalente a $(\lambda x.M)N$.

La nueva gramática de términos es:

$$M ::= x \mid \lambda x{:}\tau.M \mid MM \mid \mathsf{tt} \mid \mathsf{ff} \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M \mid \mathsf{let}\ x = M\ \mathsf{in}\ M$$

y la semántica operacional se extiende con la siguiente regla:

$$\mathsf{let}\ x = N\ \mathsf{in}\ M \to M\{x := N\}$$

Notar que $\mathsf{let}\ x = N\ \mathsf{in}\ M$ reduce exactamente igual que $(\lambda x.M)N$.
Finalmente, las reglas de tipado simple de la Definición 1.25 se extienden con la siguiente regla:

$$\frac{\Gamma \vdash M : \tau \quad \Gamma, x : \tau \vdash N : \sigma}{\Gamma \vdash \mathsf{let}\ x = N\ \mathsf{in}\ M : \sigma}\ \mathsf{let}$$

Ejercicio 2.6. Mostrar que la regla de tipado del let se puede obtener desde la derivación de $(\lambda x.M)N$.

La ideal de polimorfismo let es que sólo vamos a permitir $\forall$, en la variable ligada por el let. La variable ligada por la abstracción, no permite tipos polimórficos.
Tenemos que distinguir tipos sin cuantificadores (los que llamaremos simplemente "tipos") de tipos cuantificados (que llamaremos "esquemas de tipos").

Definición 2.7 Un esquema de tipo tiene forma $\forall X_1 \ldots \forall X_n.[\tau]$, donde τ es un tipo, con $n \geq 0$.

Definimos entonces una gramática a dos niveles: uno para tipos y otro para esquemas de tipos:

$$\begin{aligned} \tau &::= X \mid \mathtt{Bool} \mid \tau \Rightarrow \tau \\ e &::= [\tau] \mid \forall X.e \end{aligned}$$

Si τ es un tipo, $[\tau]$ es un esquema de tipo formado por el tipo τ donde ninguna variable está cuantificada.

Definición 2.8 Ahora que tenemos variables y un ligador ($\forall$) en los tipos, extendemos la definición de variables libres (FV) para esquemas tipos:

$$\begin{aligned} \mathsf{FV}([X]) &= \{X\} \\ \mathsf{FV}([\mathtt{Bool}]) &= \emptyset \\ \mathsf{FV}([\tau \Rightarrow \sigma]) &= \mathsf{FV}([\tau]) \cup \mathsf{FV}([\sigma]) \\ \mathsf{FV}(\forall X.e) &= \mathsf{FV}(e) \setminus \{X\} \end{aligned}$$

Los contextos ahora dan un esquema de tipo a cada variable de término.

Primera aproximación: Damos un primer sistema de tipos para polimorfismo let, que no es dirigido por sintaxis, pero cumple con que sólo se puede cuantificar la variable del let.

Definición 2.9 (Polimorfismo let) El sistema de tipos asocia contextos y términos con esquemas de tipos, $\Gamma \vdash M : e$, y viene dado por

$$\frac{}{\Gamma, x : e \vdash x : e}\ ax_v \qquad \frac{\Gamma, x : [\tau] \vdash M : [\sigma]}{\Gamma \vdash \lambda x{:}[\tau].M : [\tau \to \sigma]}\ \to_i \qquad \frac{\Gamma \vdash M : [\tau \to \sigma] \quad \Gamma \vdash N : [\tau]}{\Gamma \vdash MN : [\sigma]}\ \to_e$$

$$\frac{}{\Gamma \vdash \mathsf{tt} : [\mathtt{Bool}]}\ ax_t \qquad \frac{}{\Gamma \vdash \mathsf{ff} : [\mathtt{Bool}]}\ ax_f \qquad \frac{\Gamma \vdash M : [\mathtt{Bool}] \quad \Gamma \vdash N_1 : [\tau] \quad \Gamma \vdash N_2 : [\tau]}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : [\tau]}\ \mathtt{if}$$

$$\frac{\Gamma \vdash N : e \quad \Gamma, x : e \vdash M : [\tau]}{\Gamma \vdash \mathsf{let}\ x = N\ \mathsf{in}\ M : [\tau]}\ \mathsf{let} \qquad \frac{\Gamma \vdash M : e \quad X \notin \mathsf{FV}(\Gamma)}{\Gamma \vdash M : \forall X.e}\ \forall_i \qquad \frac{\Gamma \vdash M : \forall X.e}{\Gamma \vdash M : e\{X := \tau\}}\ \forall_e$$

En la definición anterior se atribuye un esquema de tipo a cada término, en particular a las variables. La regla $\to_i$ pide $x : [\tau]$, es decir, un esquema sin cuantificar. Sólo let permite darle a la variable cualquier esquema.

Ejercicio 2.10. Tipar los siguientes términos con tipos polimórficos, eligiendo convenientemente los esquemas de tipos e e'.

1. $\lambda x{:}e.x$
2. $\mathsf{let}\ i = \lambda x{:}e.x\ \mathsf{in}\ ii$
3. $(\lambda f{:}e.ff)(\lambda x{:}e'.x)$
4. $(\lambda x{:}e.xx)(\lambda x{:}e'.xx)$

Sistema dirigido por sintaxis de Hindley-Milner: La idea es que la inferencia de el tipos sea decidible, para ello se toma un sistema de tipos en el que cada término tiene una sola regla para derivar su tipo (eso es, dirigido por sintaxis, como teníamos en tipos simples).

Definición 2.11 (Relación de orden entre esquemas de tipos) La relación $\preceq$ es una relación de orden entre esquemas de tipos definida por las siguientes reglas.

$$e \preceq \forall X.e, \text{ si } X \notin \mathsf{FV}(e) \qquad \text{y} \qquad \forall X.e \preceq e\{X := \tau\}$$

Definición 2.12 (Cierre de un esquema de tipo respecto a un contexto) Sea Γ un contexto, τ un tipo, y $\vec{X} = \mathsf{FV}(\tau) \setminus \mathsf{FV}(\Gamma)$. Definimos el cierre de τ respecto a Γ como $\vec{\Gamma}(\tau) = \forall \vec{X}.[\tau]$.

Ejemplo 2.13

- El cierre de `Bool` respecto a cualquier Γ es $\vec{\Gamma}(\mathtt{Bool}) = [\mathtt{Bool}]$.

- El cierre de $X \Rightarrow Y \Rightarrow Z$ respecto a $\Gamma = x : X, w : W$ es

$$\{x : X, \vec{w} : W\}(X \Rightarrow Y \Rightarrow Z) = \forall Y.\forall Z.[X \Rightarrow Y \Rightarrow Z]$$

Definición 2.14 (Sistema de tipos de Hindley-Milner)

$$\frac{e \preceq e'}{\Gamma, x : e \vdash x : e'}\ ax_v \qquad \frac{\Gamma, x : [\tau] \vdash M : [\sigma]}{\Gamma \vdash \lambda x{:}[\tau].M : [\tau \to \sigma]}\ \to_i \qquad \frac{\Gamma \vdash M : [\tau \to \sigma] \quad \Gamma \vdash N : [\tau]}{\Gamma \vdash MN : [\sigma]}\ \to_e$$

$$\frac{}{\Gamma \vdash \mathbf{tt} : [\mathtt{Bool}]}\ ax_t \qquad \frac{}{\Gamma \vdash \mathbf{ff} : [\mathtt{Bool}]}\ ax_f \qquad \frac{\Gamma \vdash M : [\mathtt{Bool}] \quad \Gamma \vdash N_1 : [\tau] \quad \Gamma \vdash N_2 : [\tau]}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 : [\tau]}\ \mathtt{if}$$

$$\frac{\Gamma \vdash N : [\tau] \quad \Gamma, x : \vec{\Gamma}(\tau) \vdash M : [\sigma]}{\Gamma \vdash \mathsf{let}\ x = N\ \mathsf{in}\ M : [\sigma]}\ \mathsf{let}$$

Observación. Si permitimos $\forall$ en λ obtenemos el Sistema F de la Definición 2.4, pero la tipabilidad es indecidible (teorema de Wells), es decir, está demostrado que no existe algoritmo de inferencia.
Por eso, dar polimorfismo sólo a let es un buen compromiso que permite reusabilidad de código e inferencia de tipos.

2.2. Recursión

Para esta sección, vamos a extender el cálculo lambda con números naturales y operaciones aritméticas, para tener ejemplos más interesantes.

Definición 2.15 (Cálculo lambda con booleanos y números naturales) La gramática de términos es la siguiente:

$$\begin{aligned} M ::= {} & x \mid \lambda x.M \mid MM \\ & \mid \mathbf{tt} \mid \mathbf{ff} \mid \mathsf{if}\ M\ \mathsf{then}\ M\ \mathsf{else}\ M \\ & \mid \mathsf{zero} \mid \mathsf{succ}(M) \mid \mathsf{pred}(M) \mid \mathsf{isZero}(M) \mid M \times M \end{aligned}$$

Usamos $\underline{n}$ como abreviación de $\mathsf{succ}(\ldots\mathsf{succ}(\mathsf{zero}))$ con n sucesores.
Las reglas de la semántica operacional son las siguientes.

$$\begin{aligned} (\lambda x.M)N &\to M\{x := N\} && (\beta\text{ reducción}) \\ \mathsf{if}\ \mathbf{tt}\ \mathsf{then}\ M\ \mathsf{else}\ N &\to M && (\mathrm{if}_1) \\ \mathsf{if}\ \mathbf{ff}\ \mathsf{then}\ M\ \mathsf{else}\ N &\to N && (\mathrm{if}_2) \\ \mathsf{pred}(\mathsf{succ}(M)) &\to M && (\mathsf{pred}) \\ \mathsf{isZero}(\mathsf{zero}) &\to \mathbf{tt} && (\mathsf{isZero}_0) \\ \mathsf{isZero}(\mathsf{succ}(M)) &\to \mathbf{ff} && (\mathsf{isZero}_n) \\ \underline{m} \times \underline{n} &\to \underline{m \times n} && (\times) \end{aligned}$$

y las reglas de congruencia correspondientes, que se dejan como ejercicio.

Ejemplo 2.16 $\lambda x.\mathsf{succ}(x)$ es la función que suma uno, por ejemplo

$$(\lambda x.\mathsf{succ}(x))\underline{3} \to \mathsf{succ}(\underline{3}) = \mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathsf{succ}(\mathsf{zero})))) = \underline{4}$$

¿Cómo escribiríamos la función factorial?

$$\lambda x.\mathsf{if}\ \mathsf{isZero}(x)\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ x \times (\underbrace{Fact}_{?}(\mathsf{pred}(x)))$$

Vamos a extender nuevamente el cálculo con con un símbolo (palabra clave) "μ" que liga una variable en su argumento tal que $\mu f.M$ es el punto fijo[2] de $\lambda f.M$.

$$\mu f.M \text{ se comporta como } (\lambda f.M)(\mu f.M)$$

Por lo tanto,

$$Fact = \mu f.\underbrace{\lambda n.\mathsf{if}\ \mathsf{isZero}(n)\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ n \times (f\ \mathsf{pred}(n))}_{M}$$

Ejemplo 2.17

$$\begin{aligned}
Fact\ \underline{2} &= (\mu f.M)\underline{2} \\
&\to (\lambda n.\mathsf{if}\ \mathsf{isZero}(n)\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ n \times Fact\ \mathsf{pred}(n))\underline{2} \\
&\to \mathsf{if}\ \mathsf{isZero}(\underline{2})\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ \underline{2} \times Fact\ \mathsf{pred}(\underline{2}) \\
&\to \mathsf{if}\ \mathbf{ff}\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ \underline{2} \times Fact\ \mathsf{pred}(\underline{2}) \\
&\to \underline{2} \times Fact\ \mathsf{pred}(\underline{2}) \\
&\to \underline{2} \times Fact\ \underline{1} \\
&= \underline{2} \times (\mu f.M)\underline{1} \\
&\to \underline{2} \times (\lambda n.\mathsf{if}\ \mathsf{isZero}(n)\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ n \times Fact\ \mathsf{pred}(n))\underline{1} \\
&\to \underline{2} \times \mathsf{if}\ \mathsf{isZero}(\underline{1})\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ \underline{1} \times Fact\ \mathsf{pred}(\underline{1}) \\
&\to \underline{2} \times \mathsf{if}\ \mathbf{ff}\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ \underline{1} \times Fact\ \mathsf{pred}(\underline{1}) \\
&\to \underline{2} \times (\underline{1} \times Fact\ \mathsf{pred}(\underline{1})) \\
&= \underline{2} \times (\underline{1} \times (\mu f.M)\mathsf{pred}(\underline{1})) \\
&= \underline{2} \times (\underline{1} \times (\mu f.M)\mathsf{zero}) \\
&\to \underline{2} \times (\underline{1} \times (\lambda n.\mathsf{if}\ \mathsf{isZero}(n)\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ n \times Fact\ \mathsf{pred}(n))\mathsf{zero}) \\
&\to \underline{2} \times (\underline{1} \times (\mathsf{if}\ \mathsf{isZero}(\mathsf{zero})\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ \mathsf{zero} \times Fact\ \mathsf{pred}(\mathsf{zero}))) \\
&\to \underline{2} \times (\underline{1} \times (\mathsf{if}\ \mathbf{tt}\ \mathsf{then}\ \underline{1}\ \mathsf{else}\ \mathsf{zero} \times Fact\ \mathsf{pred}(\mathsf{zero}))) \\
&\to \underline{2} \times (\underline{1} \times \underline{1}) \\
&\to \underline{2} \times \underline{1} \\
&\to \underline{2}
\end{aligned}$$

2.2.1. No terminación

Ejemplo 2.18

$$\mu x.x \to x\{x := \mu x.x\} = \mu x.x$$

¡Es Ω!

[2] x es el punto fijo de f si y solo si $f(x) = x$.

Ejercicio 2.19. ¿El término $(\mu f.\lambda x.fx)0$ termina?

Fix sin μ

$$Y = \lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$$

Sea F una función cualquiera:

$$YF \to (\lambda x.F(xx))(\lambda x.F(xx)) \to F(YF)$$

¡YF es el punto fijo de F!

$$\underbrace{\mu f.M}_{YF} \to (\underbrace{\lambda f.M}_{F})(\underbrace{\mu f.M}_{YF})$$

Ejercicio 2.20. Escribir el factorial sin usar μ.

2.2.2. Tipado del punto fijo

Podemos tipar el punto fijo agregando la siguiente regla.

$$\frac{\Gamma, x : \tau \vdash M : \tau}{\Gamma \vdash \mu x.M : \tau}\ \mathsf{fix}$$

Ejercicio 2.21. Tipar el factorial.

3. Clase 3

3.1. Interpretación

En esta sección voy a poner los ejemplos motivadores con números naturales, pero las definiciones serán para cálculo lambda con booleanos y punto fijo. La extensión de estos resultados a números naturales se verá en práctica, y también la pueden encontrar en el libro de Dowek y Lévy (ver Bibliografía en página 1).

El programa que calcula el valor de un término se llama intérprete.

3.1.1. Interpretación en CBN

Supongamos que queremos un programa que tome un término de cálculo lambda y nos devuelva el valor al que llega. Por ejemplo, que tome $(\lambda x.x \times x)(2+2)$ y nos devuelva 16.
El programa debe reemplazar todas las x por $(2+2)$ para obtener $(2+2)\times(2+2)$, lo cual puede ser muy costoso en tiempo. Una alternativa es guardar $x = 2+2$ en una estructura anexa llamada contexto y evaluar $x \times x$ en ese contexto.
En un contexto permitimos tener la misma variable varias veces, y se dará la prioridad a la de más a la derecha.

Ejemplo 3.1 En el contexto $x = 3, y = 4, x = 5, z = 8$, x vale 5 y no 3.

Evaluamos términos con variables libres y cuando queremos evaluar la variable en sí, la buscamos en el contexto.
Si el término inicial es cerrado, cada vez que busquemos una variable, esta estará en el contexto.

Si en el contexto encontramos $x = M$, donde M no es un valor, deberíamos encontrar también el contexto en el que M debe ser evaluado.

Definición 3.2 (Thunk) Un par $\langle M, \Gamma\rangle$ formado por un término y un contexto de evaluación, se llama thunk.

Definición 3.3 (Cierre) Cuando queremos en cambio evaluar un término $\lambda x.M$ en un contexto, el resultado no puede ser simplemente $\lambda x.M$, tiene que ser $\lambda x.M$ pero con $\lambda x.M$ "cerrado". Introducimos para esto otro valor llamado cierre que se compone de una variable x, un término M y un contexto Γ, y se nota $\langle x, M, \Gamma\rangle$.

Definición 3.4 (Relación $\hookrightarrow$ en CBN) Vamos a definir la relación $\Gamma \vdash M \hookrightarrow V$ que se lee "M se interpreta a V en Γ".

$$\frac{\Gamma' \vdash M \hookrightarrow V}{\Gamma, x = \langle M, \Gamma'\rangle, \Delta \vdash x \hookrightarrow V}\ x \notin \mathsf{D}(\Delta)$$

$$\frac{}{\Gamma \vdash \lambda x.M \hookrightarrow \langle x, M, \Gamma\rangle} \qquad \frac{\Gamma \vdash M \hookrightarrow \langle x, M', \Gamma'\rangle \quad \Gamma', x = \langle N, \Gamma\rangle \vdash M' \hookrightarrow V}{\Gamma \vdash MN \hookrightarrow V}$$

$$\frac{}{\Gamma \vdash \mathbf{tt} \hookrightarrow \mathbf{tt}} \qquad \frac{}{\Gamma \vdash \mathbf{ff} \hookrightarrow \mathbf{ff}}$$

$$\frac{\Gamma \vdash M \hookrightarrow \mathbf{tt} \quad \Gamma \vdash N_1 \hookrightarrow V}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \hookrightarrow V} \qquad \frac{\Gamma \vdash M \hookrightarrow \mathbf{ff} \quad \Gamma \vdash N_2 \hookrightarrow V}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \hookrightarrow V}$$

$$\frac{\Gamma, x = \langle \mu x.M, \Gamma\rangle \vdash M \hookrightarrow V}{\Gamma \vdash \mu x.M \hookrightarrow V}$$

Ejemplo 3.5

$$\frac{\dfrac{}{\vdash \lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt} \hookrightarrow \langle x, \mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt}, \emptyset\rangle} \quad \dfrac{\dfrac{\dfrac{}{\vdash \mathbf{tt} \hookrightarrow \mathbf{tt}}}{x = \langle \mathbf{tt}, \emptyset\rangle \vdash x \hookrightarrow \mathbf{tt}} \quad \dfrac{}{\vdash \mathbf{ff} \hookrightarrow \mathbf{ff}}}{x = \langle \mathbf{tt}, \emptyset\rangle \vdash \mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt} \hookrightarrow \mathbf{ff}}}{\vdash (\lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt})\mathbf{tt} \hookrightarrow \mathbf{ff}}$$

3.1.2. Interpretación en CBV

En call-by-value es más fácil, ya que siempre se interpretan primero los argumentos, así que, por ejemplo, en lugar de

$$\frac{\Gamma' \vdash M \hookrightarrow V}{\Gamma, x = \langle M, \Gamma' \rangle, \Delta \vdash x \hookrightarrow V}\ x \notin \mathsf{D}(\Delta) \quad \text{tenemos} \quad \frac{}{\Gamma, x = V, \Delta \vdash x \hookrightarrow V}\ x \notin \mathsf{D}(\Delta)$$

Así que los contextos ligan variables con valores, no con thunks. Sin embargo, la regla para el punto fijo pide substituir la variable por una expresión, no por un valor, y si evaluamos esto antes de introducirlo al contexto, el cálculo no termina.

Por lo tanto el contexto contendrá valores extendidos que son o bien valores o bien thunks formados por un término $\mu x.M$ y un contexto.

Definición 3.6 (Relación $\hookrightarrow$ en CBV)

$$\frac{}{\Gamma, x = V, \Delta \vdash x \hookrightarrow V}\ x \notin \mathsf{D}(\Delta) \qquad \frac{\Gamma' \vdash \mu y.M \to V}{\Gamma, x = \langle \mu y.M, \Gamma \rangle', \Delta \vdash x \hookrightarrow V}\ x \notin \mathsf{D}(\Delta)$$

$$\frac{}{\Gamma \vdash \lambda x.M \hookrightarrow \langle x, M, \Gamma \rangle} \qquad \frac{\Gamma \vdash N \hookrightarrow W \quad \Gamma \vdash M \hookrightarrow \langle x, M', \Gamma' \rangle \quad \Gamma', x = W \vdash M' \hookrightarrow V}{\Gamma \vdash MN \hookrightarrow V}$$

$$\frac{}{\Gamma \vdash \mathbf{tt} \hookrightarrow \mathbf{tt}} \qquad \frac{}{\Gamma \vdash \mathbf{ff} \hookrightarrow \mathbf{ff}}$$

$$\frac{\Gamma \vdash M \hookrightarrow \mathbf{tt} \quad \Gamma \vdash N_1 \hookrightarrow V}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \hookrightarrow V} \qquad \frac{\Gamma \vdash M \hookrightarrow \mathbf{ff} \quad \Gamma \vdash N_2 \hookrightarrow V}{\Gamma \vdash \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2 \hookrightarrow V}$$

$$\frac{\Gamma, x = \langle \mu x.M, \Gamma \rangle \vdash M \hookrightarrow V}{\Gamma \vdash \mu x.M \hookrightarrow V}$$

Ejemplo 3.7 Retomando el Ejemplo 3.5, ahora con CBV:

$$\frac{\dfrac{}{\vdash \mathbf{tt} \hookrightarrow \mathbf{tt}} \quad \dfrac{}{\vdash \lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt} \hookrightarrow \langle x, \mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt}, \emptyset \rangle} \quad \dfrac{\dfrac{}{x = \mathbf{tt} \vdash x \hookrightarrow \mathbf{tt}} \quad \dfrac{}{\vdash \mathbf{ff} \hookrightarrow \mathbf{ff}}}{x = \mathbf{tt} \vdash \mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt} \hookrightarrow \mathbf{ff}}}{\vdash (\lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt})\mathbf{tt} \hookrightarrow \mathbf{ff}}$$

3.2. Semántica denotacional

3.2.1. Primeras definiciones

Sintaxis (o gramática) : Cómo escribir los términos. Cuáles son válidos y cuáles no.

Semántica: Qué significan.

Definición 3.8 (Semántica) La semántica de un lenguaje es una relación $\hookrightarrow$ que a cada expresión le asocia algo que le da significado.

Semántica denotacional (en programas deterministas). Para cada programa P, la relación entre las entradas y las salidas de P es una función que escribimos $[\![P]\!]$. La relación se define entonces como

$$P, E \hookrightarrow S \iff [\![P]\!]E = S$$

La pregunta es, obviamente, cómo definir $[\![P]\!]$.

Semántica operacional a grandes pasos Consiste en dar una definición inductiva de $\hookrightarrow$ que nos relacione un término con su valor. Por ejemplo, el intérprete de la Sección 3.1.

Ejemplo 3.9

$$\underbrace{(\lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt})\mathbf{tt}}_{\text{Significado de esta expresión: } \mathbf{ff}} \hookrightarrow \mathbf{ff}$$

En ese ejemplo damos semántica de acuerdo a lo que calcula.
Así, si considero

$$(\lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt})\mathbf{tt} \quad \text{y} \quad (\lambda x.x)\mathbf{ff}$$

puedo ver que los dos programas tienen la misma semántica, ante cualquier entrada produce la misma salida. Pero este es un ejemplo un tanto extremo, ya que en este caso, el programa no tiene entradas. Otro ejemplo podría ser $\mathsf{not} := \lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt}$, donde tenemos

$$\mathsf{not} \hookrightarrow \langle x, \mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt}, \emptyset\rangle$$

Aquí vemos claramente que este programa no tiene la misma semántica que $\mathsf{id} := \lambda x.x$, ya que $\mathsf{id} \hookrightarrow \langle x, x, \emptyset\rangle$, y tenemos, por ejemplo, que not produce la salida es **ff** cuando la entrada es **tt**, mientras id es al revés.

Semántica operacional a pequeños pasos También llamada semántica por reescritura. Consiste en definir $\hookrightarrow$ a partir de otra relación $\to$ que describe las etapas elementales. Ejemplo:

$$(\lambda x.\mathsf{if}\ x\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt})\mathbf{tt} \to \mathsf{if}\ \mathbf{tt}\ \mathsf{then}\ \mathbf{ff}\ \mathsf{else}\ \mathbf{tt}$$

$$M \hookrightarrow V \iff M \to^* V \qquad \text{y } V \text{ irreducible}$$

donde $\to^*$ es la clausura reflexiva y transitiva de $\to$.

Resultados de un programa. Un programa puede

- dar un resultado;
- producir un error; o
- no terminar.

Los errores se pueden considerar como resultados particulares.
Para expresar programas que no terminan hay varias formas de expresar su semántica:
La primera consiste en considerar que si M no termina, entonces no existe V tal que $M \hookrightarrow V$. La segunda consiste en agregar un elemento particular $\bot$ a los valores de salida y considerar que si M no termina, entonces $M \hookrightarrow \bot$.
En el cálculo lambda tipado con booleanos y punto fijo, todo término cerrado que no contenga μ, termina en un valor. Esta propiedad es la combinación de dos propiedades:

1. Todo término bien tipado que no contenga μ, termina (Teorema 1.32).
2. Todo término cerrado irreducible y bien tipado, es un valor. A este teorema se lo conoce como teorema de progreso.

Ejercicio 3.10. Probar el teorema de progreso del item 2, por inducción estructural en los términos.

Es decir, en en cálculo lambda con booleanos, un programa o bien da un resultado, o bien no termina.

A fin de considerar también programas que producen error, vamos a utilizar la extensión con números naturales de la Definición 2.15, en donde consideramos que $\mathsf{pred}(0)$, es un error, ya que no reduce, ni es un valor.

3.2.2. La semántica denotacional de cálculo lambda tipado

La semántica denotacional consiste en dar significado a un programa, construyendo un objeto matemático, llamado denotación.
En general, en los lenguajes funcionales buscamos reducir la distancia que separa la noción de programa de la de la función que implementa. Es decir, se busca reducir la distancia entre un programa y una semántica denotacional.

Interpretación de los tipos. A cada tipo le asociamos un conjunto:

$$\begin{aligned} [\![\mathtt{Bool}]\!] &= \mathbb{B} \\ [\![\mathtt{Nat}]\!] &= \mathbb{N} \\ [\![\tau \to \sigma]\!] &= [\![\tau]\!] \to [\![\sigma]\!] \end{aligned}$$

donde $[\![\tau]\!] \to [\![\sigma]\!]$ es el conjunto de funciones de $[\![\tau]\!]$ en $[\![\sigma]\!]$.

Interpretación de los términos. A cada término M de tipo τ le asociamos un elemento $[\![M]\!]$ del conjunto $[\![\tau]\!]$. Si M tiene variables libres, necesitamos dar una función que a cada $x : \sigma$ en el contexto Γ, le asigne un elemento $V \in [\![\tau]\!]$. A dicha función le llamamos valuación, y la notamos v. Una valuación v se dice válida para un contexto Γ si para toda variable x tal que $x : \tau \in \Gamma$, $v(x) \in [\![\tau]\!]$.

$$\begin{aligned} [\![x]\!]_v &= v(x) \\ [\![\lambda x{:}\tau.M]\!]_v &= V^{[\![\tau]\!]} \mapsto [\![M]\!]_{v,x=V} \\ [\![MN]\!]_v &= [\![M]\!]_v [\![N]\!]_v \\ [\![\mathbf{tt}]\!]_v &= \mathsf{true} \\ [\![\mathbf{ff}]\!]_v &= \mathsf{false} \\ [\![\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ O]\!]_v &= \begin{cases} [\![N]\!]_v & \text{si } [\![M]\!]_v = \mathsf{true} \\ [\![O]\!]_v & \text{sino} \end{cases} \\ [\![0]\!]_v &= 0 \\ [\![\mathsf{succ}(M)]\!]_v &= [\![M]\!]_v + 1 \\ [\![\mathsf{pred}(M)]\!]_v &= [\![M]\!]_v - 1 \end{aligned}$$

$$[\![\mathsf{isZero}(M)]\!]_v = \begin{cases} \mathsf{true} & \text{si } [\![M]\!]_v = 0 \\ \mathsf{false} & \text{sino} \end{cases}$$

Hasta ahí es trivial: un programa es una función y su semántica es la misma función. Esta trivialidad es uno de los objetivos de los lenguajes funcionales.

Observaciones.

1. $\mathsf{pred}(0)$ es un error en cálculo lambda (no reduce) y no está definido en matemática (ya que trabajamos sólo con los naturales). Para que esta definición sea correcta hay que agregar un elemento error a cada conjunto $[\![\tau]\!]$ y adaptar la definición

$$[\![\mathsf{pred}(M)]\!]_v = \begin{cases} \mathsf{error} & \text{si } [\![M]\!]_v = 0 \\ \mathsf{error} & \text{si } [\![M]\!]_v = \mathsf{error} \\ [\![M]\!]_v - 1 & \text{sino} \end{cases}$$

2. Aún no dijimos como interpretar μ.

Ejercicio 3.11. Dar la semántica de cálculo lambda con números naturales y booleanos (sin μ), propagando el error donde sea necesario.

La construcción μ es la única donde la semántica denotacional es interesante, porque es la única que se aleja de matemática, respecto a la definición de funciones:
En matemática, contrariamente a cálculo lambda, sólo podemos tomar un punto fijo si este existe y si hay varios, ¡tenemos que especificar cual!

Ejemplo 3.12

1. $x^{[\![\mathbb{N}]\!]} \mapsto x + 1$ no tiene punto fijo. Pero en cálculo lambda $\mu x{:}\mathtt{Nat}.\mathsf{succ}(x)$ es válido.
2. $f^{[\![\mathbb{N}\to\mathbb{N}]\!]} \mapsto x^{[\![\mathbb{N}]\!]} \mapsto (fx) + 1$ tampoco tiene punto fijo..., pero el término $\mu f : \mathtt{Nat} \to \mathtt{Nat}.\lambda x{:}\mathtt{Nat}.\mathsf{succ}(fx)$ también es válido.
3. $x^{[\![\mathbb{N}]\!]} \mapsto x$ tiene infinitos puntos fijos, sin embargo puedo escribir $\mu x{:}\mathtt{Nat}.x$.

Teorema 3.13 Si tomamos el punto fijo de una función que no tiene punto fijo o que tiene más de uno, el programa que obtenemos no termina. □

Observación. También existen programas con un sólo punto fijo y que no terminan en cálculo lambda. Por ejemplo $x^{[\![\mathbb{N}]\!]} \mapsto x + x$ tiene un único punto fijo (0), y sin embargo $\mu x{:}\mathtt{Nat}.x + x$ no termina.

Para comprender la semántica denotacional del punto fijo necesitamos comprender la semántica de los términos que no terminan.

- La semántica operacional a pequeños pasos no atribuye ningún resultado a estos términos.
- La semántica operacional a grandes pasos tampoco, pero podemos completar la relación $\hookrightarrow$ agregando $\bot$ tal que $\mu x{:}\mathtt{Nat}.(\mathsf{succ}(x)) \hookrightarrow \bot$.

- En la semántica denotacional la idea es hacer lo mismo: definir una función parcial $[\![\cdot]\!]$ tal que $[\![\mu x{:}\mathtt{Nat}.\mathsf{succ}(x)]\!]$ no esté definido, y adjuntamos un valor $\bot$ a $[\![\mathtt{Nat}]\!]$ tal que

$$[\![\mu x{:}\mathtt{Nat}.\mathsf{succ}(x)]\!] = \bot$$

Agregando el valor $\bot$, la interpretación del predecesor, por ejemplo, quedaría así:

$$[\![\mathsf{pred}(M)]\!]_v = \begin{cases} \mathsf{error} & \text{si } [\![M]\!]_v = 0 \\ \mathsf{error} & \text{si } [\![M]\!]_v = \mathsf{error} \\ \bot & \text{si } [\![M]\!]_v = \bot \\ [\![M]\!]_v - 1 & \text{sino} \end{cases}$$

Ahora vemos que la función $[\![\lambda x{:}\mathtt{Nat}.\mathsf{succ}(x)]\!]$ que no tenía punto fijo cuando $[\![\mathtt{Nat}]\!] = \mathbb{N}$, si $[\![\mathtt{Nat}]\!] = \mathbb{N} \cup \bot$ le podemos dar $\bot$ como semántica a ese término.
$[\![\lambda x{:}\mathtt{Nat}.x]\!]$ que tenía muchos puntos fijos, ahora tiene uno más: $\bot$.
$[\![\lambda x{:}\mathtt{Nat}.\mathsf{succ}(x)]\!]$ que tenía sólo un punto fijo en 0, ahora tiene 2: 0 y $\bot$, y el segundo es el que queremos atribuirle como semántica.

Definición 3.14 (Orden de Scott) $\bot$ es el elemento más chico de cualquier conjunto que lo contenga.

Definimos entonces $[\![\mu x{:}\tau.M]\!]$ como el punto fijo más chico de $[\![\lambda x{:}\tau.M]\!]$.

Semántica denotacional completa de cálculo lambda (sin error)

$$\begin{aligned}
[\![x]\!]_v &= v(x) \\
[\![\lambda x{:}\tau.M]\!]_v &= V^{[\![\tau]\!]} \mapsto [\![M]\!]_{v,x=V} \\
[\![MN]\!]_v &= [\![M]\!]_v [\![N]\!]_v \\
[\![\mathbf{tt}]\!]_v &= \mathsf{true} \\
[\![\mathbf{ff}]\!]_v &= \mathsf{false} \\
[\![\mathsf{if}\ M\ \mathsf{then}\ N\ \mathsf{else}\ O]\!]_v &= \begin{cases} [\![N]\!]_v & \text{si } [\![M]\!]_v = \mathsf{true} \\ [\![O]\!]_v & \text{si } [\![M]\!]_v = \mathsf{false} \\ \bot & \text{si } [\![M]\!]_v = \bot \end{cases} \\
[\![0]\!]_v &= 0 \\
[\![\mathsf{succ}(M)]\!]_v &= [\![M]\!]_v + 1 \\
[\![\mathsf{pred}(M)]\!]_v &= [\![M]\!]_v - 1 \\
[\![\mathsf{isZero}(M)]\!]_v &= \begin{cases} \mathsf{true} & \text{si } [\![M]\!]_v = 0 \\ \mathsf{false} & \text{si } [\![M]\!]_v = n \neq 0 \\ \bot & \text{si } [\![M]\!]_v = \bot \end{cases} \\
[\![\mu x{:}\tau.M]\!]_v &= \mathsf{FIX}(V^{[\![\tau]\!]} \mapsto [\![M]\!]_{v,x=V})
\end{aligned}$$

donde $\mathsf{FIX}(f)$ es el mínimo punto fijo de f.

Ejercicio 3.15. Dar la semántica denotacional completa de cálculo lambda con booleanos y números naturales, propagando el error donde sea necesario.

Observación. El mínimo de $[\![\tau]\!]$ es $\bot_\tau$ y el mínimo de $[\![\tau \to \sigma]\!] = \bot_{[\![\tau]\!] \to [\![\sigma]\!]}$ que es la función constante $\bot_{[\![\sigma]\!]}$.

El siguiente teorema nos dice que si un término está bien tipado, entonces su semántica denotacional es un elemento del conjunto que corresponde a su tipo.

Teorema 3.16 Si $\Gamma \vdash M : \tau$, entonces para toda valuación v válida en Γ se tiene $[\![M]\!]_v \in [\![\tau]\!]$.

Demostración. Procedemos por inducción estructural en la relación de tipado.

- $\Gamma, x : \tau \vdash x : \tau$. $[\![x]\!]_v = v(x) \in [\![\tau]\!]$.
- $\dfrac{\Gamma, x : \tau \vdash M : \sigma}{\Gamma \vdash \lambda x{:}\tau.M : \tau \to \sigma}$.

  $[\![\lambda x{:}\tau.M]\!]_v = V^{[\![\tau]\!]} \mapsto [\![M]\!]_{v,x=V}$. Por hipótesis de inducción $\forall V \in [\![\tau]\!], [\![M]\!]_{v,x=V} \in [\![\sigma]\!]$, es decir, $V^{[\![\tau]\!]} \mapsto [\![M]\!]_{v,x=V} \in [\![\tau]\!] \to [\![\sigma]\!] = [\![\tau \to \sigma]\!]$.
- $\dfrac{\Gamma \vdash M : \tau \to \sigma \quad \Gamma \vdash N : \tau}{\Gamma \vdash MN : \sigma}$

  $[\![MN]\!]_v = [\![M]\!]_v [\![N]\!]_v$. Por hipótesis de inducción $[\![M]\!]_v \in [\![\tau]\!] \to [\![\sigma]\!]$ y $[\![N]\!]_v \in [\![\sigma]\!]$, por lo tanto $[\![M]\!]_v [\![N]\!]_v \in [\![\sigma]\!]$.
- $\dfrac{\Gamma, x : \tau \vdash M : \tau}{\Gamma \vdash \mu x{:}\tau.M : \tau}$

  $[\![\mu x{:}\tau.M]\!]_v = FIX(V^{[\![\tau]\!]} \mapsto [\![M]\!]_{v,x=V})$. Por hipótesis de inducción, si v es válida en Γ, y $V \in [\![\tau]\!]$, $[\![M]\!]_{v,x=V} \in [\![\tau]\!]$. El punto fijo de una función de $[\![\tau]\!]$ en $[\![\tau]\!]$ es un elemento de $[\![\tau]\!]$. Por lo tanto, $[\![\mu x{:}\tau.M]\!]_v \in [\![\tau]\!]$.

Ejercicio 3.17. Completar los casos que faltan. □

El teorema de soundness (Teorema 3.20), nos dice que la semántica es correcta respecto a la reducción. Es decir, si un término está bien tipado y se reduce a otro término, entonces la semántica de ambos términos es la misma. Para poder demostrarlo, necesitaremos primero el siguiente lema de sustitución.

Lema 3.18 (Sustitución) $[\![M\{x := N\}]\!]_v = [\![M]\!]_{v,x=[\![N]\!]_v}$.

Demostración. Por inducción estructural en M.

- $M = x$. $[\![x\{x := N\}]\!]_v = [\![N]\!]_v = [\![x]\!]_{v,x=[\![N]\!]_v}$.
- $M = y$, $y \neq x$. $[\![y\{x := N\}]\!]_v = [\![y]\!]_v = [\![y]\!]_{v,x=[\![N]\!]_v}$.
- $M = \lambda y.P$. $[\![(\lambda y.P)\{x := N\}]\!]_v = [\![\lambda y.P\{x := N\}]\!]_v = V^{[\![\tau]\!]} \mapsto [\![P\{x := N\}]\!]_{v,y=V}$, que, por hipótesis de inducción, es igual a $V^{[\![\tau]\!]} \mapsto [\![P]\!]_{v,y=V,x=[\![N]\!]_v} = [\![\lambda y.P]\!]_{v,x=[\![N]\!]_v}$.

- $M = PQ$. $[\![(PQ)\{x := N\}]\!]_v = [\![P\{x := N\}Q\{x := N\}]\!]_v = [\![P\{x := N\}]\!]_v [\![Q\{x := N\}]\!]_v$, que, por hipótesis de inducción, es igual a $[\![P]\!]_{v,x=[\![N]\!]_v} [\![Q]\!]_{v,x=[\![N]\!]_v} [\![PQ]\!]_{v,x=[\![N]\!]_v}$.

Ejercicio 3.19. Completar los casos que faltan. □

Teorema 3.20 (Soundness) Si $\Gamma \vdash M : \tau$ y $M \to N$, entonces para toda valuación v válida en Γ se tiene $[\![M]\!]_v = [\![N]\!]_v$.

Demostración.

Ejercicio 3.21. Probar este teorema por inducción sobre la relación $\to$, usando el Lema 3.18 donde sea necesario. □

Esta semántica es sound pero no es complete, ya que el hecho de que dos términos tengan la misma semántica, no implica que reduzcan a lo mismo. Por ejemplo $[\![\lambda x{:}\tau.Mx]\!]_v = [\![M]\!]_v$, sin embargo $\lambda x{:}\tau.Mx \not\to M$ ni al revés. Esto es conocido, y la solución es agregar la regla llamada η-reducción o η-expansión, que son $\lambda x{:}\tau.Mx \not\to M$ y $M \to \lambda x{:}\tau.Mx$ respectivamente. Otra solución es probar una forma más débil de completitud, llamada adequación, que dice que si dos términos tienen la misma semántica, entonces son computacionalmente equivalentes. Sin embargo, esto va más allá de los contenidos de esta materia.