Características de Lenguajes de Programación – 2020

Alejandro Díaz-Caro (Jano) & Rafael Romero

versión: 2020.10.14

1. Sobre la materia

1.1. Contenidos y cronograma tentativo

Lunes	Tema	Miércoles	Tema
24 de agosto	Presentación & PCF	26 de agosto	PCF (cont.)
31 de agosto	Práctica	2 de septiembre	Estrategias de reducción
7 de septiembre	Práctica	9 de septiembre	PCF tipado
14 de septiembre	Práctica	16 de septiembre	Inferencia de tipos
21 de septiembre	Práctica	23 de septiembre	Polimorfismo
28 de septiembre	Práctica	*30 de septiembre*	*Consulta*
5 de octubre	**1er Parcial**	7 de octubre	Interpretación
~~12 de octubre~~	Feriado	~~14 de octubre~~	Paro CONADU
19 de octubre	Práctica	21 de octubre	Compilación
26 de octubre	Práctica	28 de octubre	Registros y objetos
2 de noviembre	Práctica	4 de noviembre	Semántica denotacional
9 de noviembre	Práctica	11 de noviembre	Subject reduction
16 de noviembre	Práctica	*18 de noviembre*	*Consulta*
~~23 de noviembre~~	Feriado	**25 de noviembre**	**2do Parcial**
30 de noviembre	*Consulta*	**2 de diciembre**	**Recuperatorio**
7 de diciembre	*Consulta*	**9 de diciembre**	**Integrador**

1.2. Bibliografía

- 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.

1.3. Condiciones de aprobación

- 75 % o más de tareas entregadas.
- Trabajo práctico y ambos parciales con nota superior o igual a 6.
- Promedio entre parciales superior o igual a 7.

Índice

2. PCF no tipado

2.1. Primeras definiciones

Programming language for computable functions (PCF)

- Dos construcciones de base:
  - Construcción explícita de función (sin nombre):
    $$\lambda x.t$$
  - Aplicación de una función a un argumento:
    $$tu$$
- Una constante para cada número natural.
- Operaciones: $+$, $-$, $\times$, $/$[1].
- Test a cero: $\mathsf{ifz}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v$.

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, $f(x,y) = x^2 + y^2$, en PCF se escribe
  $$\underbrace{\lambda x.\lambda y.x \times x + y \times y}_{F}$$
  $F3$ es una función que espera un argumento para elevarlo al cuadrado y sumarle 9.
- La aplicación asocia a la izquierda: $F34 = (F3)4$.

Recursión

¿Cómo escribiríamos la función factorial en PCF?

$$\lambda n.\mathsf{ifz}\ n\ \mathsf{then}\ 1\ \mathsf{else}\ n \times (\underbrace{Fact}_{?}(n-1))$$

En PCF hay un símbolo (palabra clave) "fix" que liga una variable en su argumento tal que $\mu f.G$ es el punto fijo[2] de $\lambda f.G$.

$$\mu f.G = (\lambda f.G)(\mu f.G)$$

Por lo tanto,

$$Fact = \mu f.\underbrace{\lambda n.\mathsf{ifz}\ n\ \mathsf{then}\ 1\ \mathsf{else}\ n \times (f(n-1))}_{G}$$

[1] Dado que sólo usamos naturales, consideraremos que $n - m = 0$ si $m > n$, y / representa la división euclidiana (con resto), donde la división por 0 da error.

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

Ejemplo 2.1.

$$\begin{aligned}
Fact\ 2 &= (\mu f.G)2 \\
&= ((\lambda f.G)Fact)2 \\
&= (\lambda n.\mathsf{ifz}\ n\ \mathsf{then}\ 1\ \mathsf{else}\ n \times Fact(n-1))2 \\
&= \mathsf{ifz}\ 2\ \mathsf{then}\ 1\ \mathsf{else}\ 2 \times Fact\ 1 \\
&= 2 \times Fact\ 1 \\
&= 2 \times (\mu f.G)1 \\
&= 2 \times ((\lambda f.G)Fact)1 \\
&= 2 \times (\lambda n.\mathsf{ifz}\ n\ \mathsf{then}\ 1\ \mathsf{else}\ n \times Fact(n-1))1 \\
&= 2 \times \mathsf{ifz}\ 1\ \mathsf{then}\ 1\ \mathsf{else}\ 1 \times Fact\ 0 \\
&= 2 \times 1 \\
&= 2
\end{aligned}$$

Let

La construcción let no es necesaria, pero será útil más adelante y por lo tanto también la vamos a considerar:

$$\mathsf{let}\ x = t\ \mathsf{in}\ u$$

es equivalente a $(\lambda x.u)t$.

2.2. Gramática de PCF

Los términos válidos de PCF los podemos describir con una gramática formal:

$$\begin{aligned}
t ::= x \mid \lambda x.t \mid tt \mid n \in \mathbb{N} \mid t+t \mid t-t \mid t \times t \mid t/t \\
\mid \mathsf{ifz}\ t\ \mathsf{then}\ t\ \mathsf{else}\ t \mid \mu x.t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t
\end{aligned}$$

PCF es Turing completo, es decir que todas las funciones de enteros computables son programables en PCF.

2.3. Semántica operacional

La gramática nos dice qué términos podemos escribir sintácticamente. La semántica operacional nos da el significado de los términos, al definir como operan.
Las siguientes reglas tienen la forma $t \to u$, y se lee "t reduce a u":

$$(\lambda x.u)t \to u[t/x] \qquad (\beta \text{ reducción})$$

$$p \otimes q \to n \qquad \text{Si } p \otimes q = n, \text{ con } \otimes = +, -, \times \text{ o } /$$

$$\mathsf{ifz}\ 0\ \mathsf{then}\ t\ \mathsf{else}\ u \to t$$

$$\mathsf{ifz}\ n\ \mathsf{then}\ t\ \mathsf{else}\ u \to u \qquad \text{Si } n \neq 0$$

$$\mu x.t \to t[\mu x.t/x]$$

$$\mathsf{let}\ x = t\ \mathsf{in}\ u \to u[t/x]$$

También tendremos reglas de congruencia que permitirán reducir un término dentro de otro:

$$\frac{t \to u}{tv \to uv} \qquad \frac{t \to u}{vt \to vu} \qquad \frac{t \to u}{\lambda x.t \to \lambda x.u} \qquad \frac{t \to u}{t \otimes v \to u \otimes v} \qquad \frac{t \to u}{v \otimes t \to v \otimes u}$$

Ejercicio: escribir las que faltan.

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

2.4. No terminación

Ejemplos 2.2. 1.

$$\mu x.x \to x[\mu x.x/x] = \mu x.x$$

2. Sin fix:

$$\Omega = (\lambda x.xx)(\lambda x.xx) \to xx[\lambda x.xx/x] = (\lambda x.xx)(\lambda x.xx) = \Omega$$

Ejercicio:

$$(\mu f.\lambda x.fx)0$$

¿Termina?

Fix sin fix

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

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!

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

Ejercicio: escribir el factorial sin usar fix.

2.5. Captura de variables

Ejercicio: Reducir los siguientes términos

1. $(\lambda x.\lambda x.x)23$
2. $(\lambda x.\lambda y.(\lambda x.x + y)x)54$
3. $\left.\begin{array}{l}\text{let } x = 4 \text{ in} \\ \quad \text{let } f = \lambda y.y + x \text{ in} \\ \quad\quad \text{let } x = 5 \text{ in} \\ \quad\quad\quad f6\end{array}\right\}$ En las primeras versiones de Lisp, este ejemplo daba 11 en lugar de 10.

Tenemos que definir precisamente qué significa $t[u/x]$. Lo definimos por inducción en t.

$$
\begin{array}{rll}
x[u/x] & = u & \\
y[u/x] & = y & \\
(\lambda x.t)[u/x] & = \lambda x.t & \\
(\lambda y.t)[u/x] & = \lambda y.t[u/x] & \text{Si } y \notin \mathsf{FV}(u) \\
(\lambda y.t)[u/x] & = \lambda z.t[z/y][u/x] & \text{Si } y \in \mathsf{FV}(u) \\
(tv)[u/x] & = t[u/x]v[u/x] & \\
n[u/x] & = n & \\
(t \otimes v)[u/x] & = t[u/x] \otimes v[u/x] & \\
(\mathsf{ifz}\ t\ \mathsf{then}\ v_1\ \mathsf{else}\ v_2)[u/x] & = \mathsf{ifz}\ t[u/x]\ \mathsf{then}\ v_1[u/x]\ \mathsf{else}\ v_2[u/x] & \\
(\mu x.t)[u/x] & = \mu x.t & \\
(\mu y.t)[u/x] & = \mu y.t[u/x] & \text{Si } y \notin \mathsf{FV}(u) \\
(\mu y.t)[u/x] & = \mu z.t[z/y][u/x] & \text{Si } y \in \mathsf{FV}(u) \\
(\mathsf{let}\ x = t\ \mathsf{in}\ v)[u/x] & = \mathsf{let}\ x = t[u/x]\ \mathsf{in}\ v & \\
(\mathsf{let}\ y = t\ \mathsf{in}\ v)[u/x] & = \mathsf{let}\ y = t[u/x]\ \mathsf{in}\ v[u/x] & \text{Si } y \notin \mathsf{FV}(u) \\
(\mathsf{let}\ y = t\ \mathsf{in}\ v)[u/x] & = \mathsf{let}\ y = t[u/x]\ \mathsf{in}\ v[z/y][u/x] & \text{Si } y \in \mathsf{FV}(u)
\end{array}
$$

Ejercicio: Definir, por inducción sobre t, $\mathsf{FV}(t)$.

Tarea: Definir, por inducción sobre t, $BV(t)$, es decir, el conjunto de variables ligadas de t ("bounded variables").

3. Estrategias de reducción

3.1. Primeras definiciones

Definición 3.1. Notamos $\to^*$ al cierre reflexivo y transitivo de $\to$. Es decir, si $t \to^* u$, entonces, $t = v_0 \to v_1 \to v_2 \to \cdots \to v_n = u$, con $n \geq 0$. Notamos $\to^+$ al cierre transitivo de $\to$. Es decir, si $t \to^* u$, $t = v_0 \to v_1 \to \cdots \to v_n = u$, con $n \geq 1$.

Ejemplo 3.2. $(\lambda x.x + 2)1 \to^* 3$ porque $(\lambda x.x + 2)1 \to 1 + 2 \to 3$.
También, $(\lambda x.x + 2)1 \to^+ 3$, ya que $(\lambda x.x + 2)1 \neq 3$.

Definición 3.3.

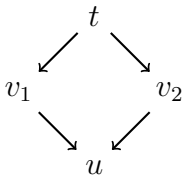
1. Un término t está en forma normal si no existe u tal que $t \to u$.

2. Un término t es normalizable (o tiene forma normal) si existe u en forma normal tal que $t \to^* u$.

3. Un término t es fuertemente normalizable si no existe una secuencia infinita v_0, v_1, ... tal que $t \to v_0 \to v_1 \to \ldots$. Es decir, toda secuencia de reducción comenzada en t debe ser finita y terminar en un término en forma normal.

Definición 3.4. Sea $\to_R$ una relación binaria, y $\to_R^*$ su cierre reflexivo y transitivo.

- $\to_R$ satisface la propiedad del diamante si $t \to_R v_1$ y $t \to_R v_2$ implica que $v_1 \to_R u$ y $v_2 \to_R u$ para algún u.



- $\to_R$ es Church-Rosser o confluente si $\to_R^*$ satisface la propiedad del diamante. Es decir, si $t \to_R^* v_1$ y $t \to_R^* v_2$ implica que $v_1 \to_R^* u$ y $v_2 \to_R^* u$ para algún u.
- $\to_R$ tiene formas normales únicas si $t \to_R^* v_1$ y $t \to_R^* v_2$, para términos en forma normal v_1 y v_2, implica $v_1 = v_2$.

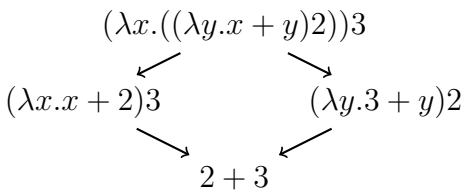
Lema 3.5.

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.

Teorema 3.6. La relación definida en la Sección 2.3 (semántica operacional de PCF) es Church-Rosser.

Ejemplo 3.7.



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

$$Fact = \mu f.\underbrace{\lambda n.\mathsf{ifz}\ n\ \mathsf{then}\ 1\ \mathsf{else}\ n \times f(n-1)}_{G} = \mu f.G$$

Entonces:

$$\begin{aligned} Fact\ 0 &= (\mu f.G)0 \\ &\to (G[Fact/f])0 \\ &\to (G[G[Fact/f]/f])0 \\ &\to (G[G[G[Fact/f]/f]/f])0 \\ &\to \cdots \to \infty \end{aligned}$$

$Fact\ 0$ tiene un único resultado, pero no cualquier camino llega a él.
Solución (en este caso): cuando hay un ifz, reducir primero el ifz antes que sus ramas. Ésto, como veremos luego, es una estrategia.
Otro ejemplo:

$$\begin{aligned} C_0 &= \lambda x.0 \\ b_1 &= (\mu f.\lambda x.fx)0 \end{aligned}$$

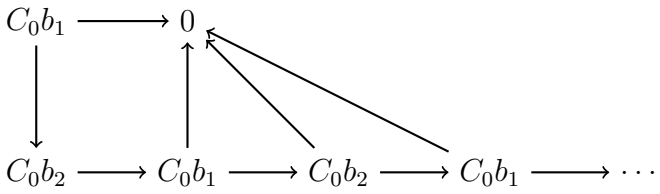
Entonces,

$$b_1 = (\mu f.\lambda x.fx)0 \to \underbrace{(\lambda x.(\mu f.\lambda x.fx)x)0}_{b_2} \to (\mu f.\lambda x.fx)0 = b_1$$

Es decir,

$$b_1 \to b_2 \to b_1 \to b_2 \to \cdots$$

Por lo tanto, tenemos el siguiente diagrama:



En Caml,

```
let rec f x = f x in
let g x = 0 in
g (f 0)
```

no termina nunca.
Mismo ejemplo en Java:

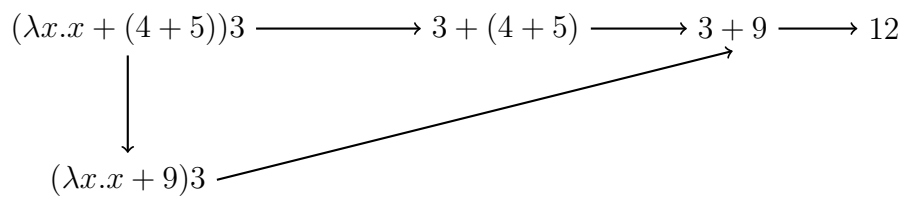
```
class Omega {
  static int f (int x) {return f(x);}
  static int g (int x) {return 0;}
  static public void main (String [] args) {
    System.out.println (g(f(0)));
  }
}
```

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

Definición 3.8. Llamamos radical (en inglés redex) a un subtérmino de un término que puede reducir.

3.2. Reducción débil

Ejemplo motivador:



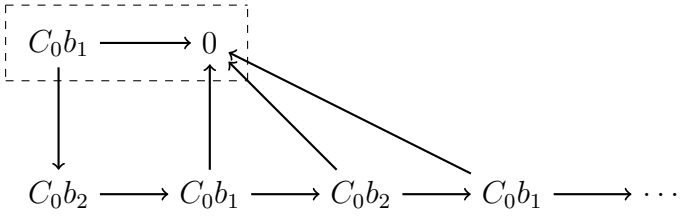
- La dirección $\to$ dice qué sucede cuando se ejecuta el programa.
- La dirección $\downarrow$ comienza a ejecutar el programa antes de recibir los argumentos, es decir, no ejecuta el programa sino que lo optimiza.

Definición 3.9. 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

$$\frac{t \to u}{\lambda x.t \to \lambda x.u}$$

3.3. Call-by-name

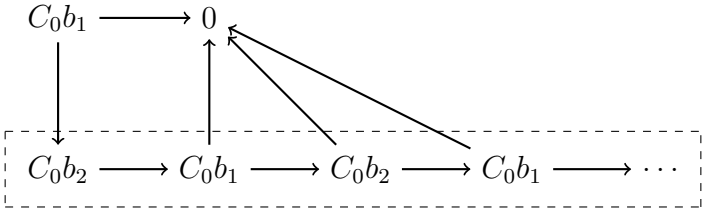


Definición 3.10. La estrategia call-by-name reduce siempre el radical más a la izquierda. En caso de ser además débil, será el más a la izquierda que no esté debajo de un λ.

Teorema 3.11 (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 $(\lambda x.0)(\mathit{Fact}\ 10)$ no necesitamos calcular el factorial de 10. Por otro lado, si tenemos $(\lambda x.x + x)(\mathit{Fact}\ 10)$, tendremos que calcular el factorial de 10 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 $(\lambda x.x + x)(\mathit{Fact}\ 10)$ reduce a $x + x$, donde x es un puntero a $\mathit{Fact}\ 10$. A eso se le llama reducción lazy.
Ejercicio: Escribir las reglas de reducción y congruencia que implementan call-by-name.

3.4. Call-by-value



Definición 3.12. A los términos t de PCF tales que $\mathsf{FV}(t) = \emptyset$ y que t esté en forma normal, se les llaman valores.

Definición 3.13. La estrategia call-by-value consiste en evaluar siempre los argumentos antes de pasarlos a la función. La idea es que

$$(\lambda x.t)v$$

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

En $(\lambda x.x + x)(\mathit{Fact}\ 10)$ comenzamos por reducir el factorial, obtenemos 3628800 y recién ahí lo pasamos a la función. De esa manera el factorial es calculado una vez.
Ejercicio: Escribir las reglas que implementan call-by-value.

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

4. PCF tipado

4.1. Introducción

Ejemplos motivadores:

$$\begin{gathered}(\lambda x.x + 1)\lambda x.x + 2 \to (\lambda x.x + 2) + 1 \\ \mathsf{ifz}\ \lambda x.x\ \mathsf{then}\ 0\ \mathsf{else}\ 1 \not\to \\ (\lambda x.x)1\lambda x.x \to 1\lambda x.x\end{gathered}$$

¡Todo es aplicable a todo! Sin restricciones. Sumar 1 a una función no tiene sentido. Hacer un ifz sobre algo que no es un número o pasarle un argumento a un número, tampoco.
Idea: detectar este tipo de errores sintácticamente. Por ejemplo:

$$\frac{\lambda x.x \text{ recive un argumento y devuelve lo mismo} \quad 1 \text{ es una constante}}{(\lambda x.x)1 \text{ es una constante}}$$

Es decir, deducimos que no tiene sentido pasarle un argumento a $(\lambda x.x)1$, ya que es una constante, y lo dedujimos sin tener que ejecutar el programa.

En matemáticas:

$$\begin{array}{ccc}\text{Función: Dominio} & \longrightarrow & \text{Codominio} \\ & \nwarrow \quad \nearrow & \\ & \text{Cualquier conjunto} & \end{array}$$

Ejemplo:

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

¿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), y poco importa si es par o no. Así que vamos a restringir las clases de conjuntos que se pueden utilizar como dominios. A estos conjuntos los llamamos **tipos**.

4.2. Gramática de PCF tipado

Definición 4.1. Los tipos de PCF los definimos inductivamente por:

- nat (es decir $\mathbb{N}$) es un tipo.
- Si A y B son tipos, $A \Rightarrow B$ es un tipo que representa las funciones de A en B.

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:

- $\lambda x : \mathsf{nat}.x$ es la identidad sobre los naturales
- $\lambda x : \mathsf{nat} \Rightarrow \mathsf{nat}.x$ es la identidad sobre las funciones de naturales en naturales.

En fix y let también es necesario marcar el tipo de la variable ligada.

La gramática de PCF tipado la definimos con una gramática de tipos y una de términos, de la siguiente manera:

$$A ::= \mathsf{nat} \mid A \Rightarrow A$$
$$t ::= x \mid \lambda x : A.t \mid tt \mid n \mid t \otimes t \mid \mathsf{ifz}\ t\ \mathsf{then}\ t\ \mathsf{else}\ t \mid \mu x : A.t \mid \mathsf{let}\ x : A = t\ \mathsf{in}\ t$$

donde $\otimes = +, -, \times, /$ y $n \in \mathbb{N}$.

4.3. La relación de tipado

Queremos definir inductivamente la relación $t : A$ (es decir, el término t tiene el tipo A). Pero si hay variables libres en t, ¿cómo las tipo?[3].
Por ejemplo:

$$\lambda x : \mathsf{nat}.yx$$

¿Qué tipo tiene y? Claramente tiene que ser una función de nat en algo, pero ¿cómo defino ese algo?

[3]Dijimos que sólo 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.

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

$$\underbrace{y : \mathsf{nat} \Rightarrow \mathsf{nat}}_{\text{contexto}} \vdash \lambda x : \mathsf{nat}.yx : \mathsf{nat} \Rightarrow \mathsf{nat}$$

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

Definición 4.2. La relación de tipado $\Gamma \vdash t : A$ se define inductivamente por:

$$\frac{}{\Gamma, x : A \vdash x : A}\ ax_v \qquad \frac{}{\Gamma \vdash n : \mathsf{nat}}\ ax_c$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \Rightarrow B}\ \Rightarrow_i \qquad \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B}\ \Rightarrow_e$$

$$\frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash u : \mathsf{nat}}{\Gamma \vdash t \otimes u : \mathsf{nat}}\ \otimes \qquad \frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash u : A \quad \Gamma \vdash v : A}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v : A}\ \mathsf{ifz}$$

$$\frac{\Gamma, x : A \vdash t : A}{\Gamma \vdash \mu x : A.t : A}\ \mathsf{fix} \qquad \frac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{let}\ x : A = u\ \mathsf{in}\ t : B}\ \mathsf{let}$$

Donde $\otimes$ son cuatro reglas, una para cada operación aritmética.

Ejemplo 4.3. Tipar $\lambda x : \mathsf{nat} \Rightarrow \mathsf{nat}.x((\lambda y : \mathsf{nat}.y + 2)3)$.
Sean $\Delta = x : \mathsf{nat} \Rightarrow \mathsf{nat}$, y $\Gamma = \Delta, y : \mathsf{nat}$. Entonces,

$$\dfrac{\dfrac{\dfrac{}{\Delta \vdash x : \mathsf{nat} \Rightarrow \mathsf{nat}}\ ax_v \quad \dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \mathsf{nat}}\ ax_v \quad \dfrac{}{\Gamma \vdash 2 : \mathsf{nat}}\ ax_c}{\Gamma \vdash y + 2 : \mathsf{nat}}\ +}{\Delta \vdash \lambda y : \mathsf{nat}.y + 2 : \mathsf{nat} \Rightarrow \mathsf{nat}}\ \Rightarrow_i \quad \dfrac{}{\Delta \vdash 3 : \mathsf{nat}}\ ax_c}{\Delta \vdash (\lambda y : \mathsf{nat}.y + 2)3 : \mathsf{nat}}\ \Rightarrow_e}{\Delta \vdash x((\lambda y : \mathsf{nat}.y + 2)3) : \mathsf{nat}}\ \Rightarrow_e}{\vdash \lambda x : \mathsf{nat} \Rightarrow \mathsf{nat}.x((\lambda y : \mathsf{nat}.y + 2)3) : (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{nat}}\ \Rightarrow_i$$

Ejercicio 1: tipar factorial.
Ejercicio 2: tipar $\lambda x : A.xx$ para algún A.

Teorema 4.4 (Subject reduction o conservación de tipos)**.**
Si $\Gamma \vdash t : A$ y $t \to u$ entonces $\Gamma \vdash u : A$.

Es decir: si deducimos el tipo A para un término (usando la Definición 4.2), o sea, sin ejecutar el programa, y luego ejecutamos el programa obteniendo u, entonces el término u 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.

Teorema 4.5 (Teorema de Tait o normalización fuerte)**.**
Todo término tipado que no contenga a fix, termina.

¿Qué sucede con $\Omega = (\lambda x.xx)(\lambda x.xx)$? No es tipable. Es decir, no existe un tipo A tal que $\vdash \Omega : A$.

Ejercicio: Extender PCF con constructores para pares: (t, u), $\mathsf{fst}(t, u)$ y $\mathsf{snd}(t, u)$. Dar la gramática, semántica operacional y reglas de tipado.

5. Inferencia de tipos simples

5.1. Introducción

En muchos lenguajes el programador debe indicar el tipo de las variables. Por ejemplo

$$\lambda x : \mathsf{nat}.x + 1$$

Sin embargo, en este caso es un poco innecesario, ya que la operación + sólo puede ocurrir entre naturales, y por lo tanto era posible inferir que x debía ser de tipo nat.
Podemos entonces liberar al programador de escribir los tipos y escribir un algoritmo que los infiera.

Estilo Curry. Escribimos variables sin tipos, como hicimos con PCF no tipado, y definimos independientemente el lenguaje de tipos, como ya lo definimos.
Así, en vez de $\vdash \lambda x : \mathsf{nat}.x + 1 : \mathsf{nat} \Rightarrow \mathsf{nat}$ escribimos $\vdash \lambda x.x + 1 : \mathsf{nat} \Rightarrow \mathsf{nat}$.

Gramática

$$A ::= \mathsf{nat} \mid A \Rightarrow A$$
$$t ::= x \mid \lambda x.t \mid tt \mid n \mid t \otimes t \mid \mathsf{ifz}\ t\ \mathsf{then}\ t\ \mathsf{else}\ t \mid \mu x.t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t$$

Reglas de tipado

$$\frac{}{\Gamma, x : A \vdash x : A}\ ax_v \qquad \frac{}{\Gamma \vdash n : \mathsf{nat}}\ ax_c$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x.t : A \Rightarrow B}\ \Rightarrow_i \qquad \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B}\ \Rightarrow_e$$

$$\frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash u : \mathsf{nat}}{\Gamma \vdash t \otimes u : \mathsf{nat}}\ \otimes \qquad \frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash u : A \quad \Gamma \vdash v : A}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v : A}\ \mathsf{ifz}$$

$$\frac{\Gamma, x : A \vdash t : A}{\Gamma \vdash \mu x.t : A}\ \mathsf{fix} \qquad \frac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{let}\ x = u\ \mathsf{in}\ t : B}\ \mathsf{let}$$

Ahora, para el mismo término, podemos derivar diferentes tipos. Por ejemplo:

$$\frac{\dfrac{}{x : \mathsf{nat} \vdash x : \mathsf{nat}}\ ax_v}{\vdash \lambda x.x : \mathsf{nat} \Rightarrow \mathsf{nat}}\ \Rightarrow_i \qquad \frac{\dfrac{}{x : \mathsf{nat} \Rightarrow \mathsf{nat} \vdash x : \mathsf{nat} \Rightarrow \mathsf{nat}}\ ax_v}{\vdash \lambda x.x : (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow (\mathsf{nat} \Rightarrow \mathsf{nat})}\ \Rightarrow_i$$

podemos simplemente decir:

$$\dfrac{\dfrac{}{x : X \vdash x : X}\, ax_v}{\vdash \lambda x.x : X \Rightarrow X} \Rightarrow_i$$

donde X es una variable en el sentido de que "desconocido", o "cualquier tipo".

Definición 5.1. Notamos $A(X)$ a un tipo cualquiera que contiene alguna variable X. Notamos θ a una substitución de meta-variables por tipos.

Ejemplo 5.2. Sea $\theta = \mathsf{nat}/X, \mathsf{nat} \Rightarrow \mathsf{nat}/Y$. Entonces,

$$\theta(X \Rightarrow Y) = \mathsf{nat} \Rightarrow \mathsf{nat} \Rightarrow \mathsf{nat}$$

Teorema 5.3. Si $\vdash t : A(X)$, entonces $\vdash t : \theta A(X)$ para cualquier substitución θ.

Ejemplo 5.4. Como vimos anteriormente, $\vdash \lambda x.x : X \Rightarrow X$. Por lo tanto, tomando $\theta = \mathsf{nat}/X$,

$$\vdash \lambda x.x : \mathsf{nat} \Rightarrow \mathsf{nat}$$

Tomando $\theta = (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{nat}/X$ tenemos

$$\vdash \lambda x.x : ((\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{nat}) \Rightarrow ((\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{nat})$$

5.2. Algoritmo de Hindley

Supongamos que queremos tipar $\lambda f.2 + f1$, entonces ponemos en el contexto $f : X$ y tenemos que llegar a $f : X \vdash 2 + f1 : Y$ para algún X e Y. Dado que hay una suma, tenemos que tipar $f : X \vdash 2 : \mathsf{nat}$ y $f : X \vdash f1 : \mathsf{nat}$, y como la suma sólo puede ser nat, sabemos que $Y = \mathsf{nat}$. Por lo tanto tenemos $f : X \vdash 2 : \mathsf{nat}$ y $f : X \vdash f1 : \mathsf{nat}$. Entonces, como es una aplicación, tenemos que $f : X \vdash f : Z \Rightarrow \mathsf{nat}$ y $f : X \vdash 1 : Z$. Entonces $Z = \mathsf{nat}$ y $X = \mathsf{nat} \Rightarrow \mathsf{nat}$. Por lo tanto:

$$\dfrac{\dfrac{\dfrac{}{f : \mathsf{nat} \Rightarrow \mathsf{nat} \vdash 2 : \mathsf{nat}}\, ax_c \quad \dfrac{\dfrac{}{f : \mathsf{nat} \Rightarrow \mathsf{nat} \vdash f : \mathsf{nat} \Rightarrow \mathsf{nat}}\, ax_v \quad \dfrac{}{f : \mathsf{nat} \Rightarrow \mathsf{nat} \vdash 1 : \mathsf{nat}}\, ax_c}{f : \mathsf{nat} \Rightarrow \mathsf{nat} \vdash f1 : \mathsf{nat}} \Rightarrow_e}{f : \mathsf{nat} \Rightarrow \mathsf{nat} \vdash 2 + f1 : \mathsf{nat}} +}{\vdash \lambda f.2 + f1 : \mathsf{nat} \Rightarrow \mathsf{nat}} \Rightarrow_i$$

Es decir, que en este caso hay una única solución.

Vamos a describir la primera parte del algoritmo inductivamente como una relación

$$\Gamma \vdash t \rightsquigarrow A, \tau$$

entre un contexto Γ y un término t, con un tipo A y un conjunto de ecuaciones τ.

$$\dfrac{}{\Gamma, x : X \vdash x \rightsquigarrow X, \emptyset} \qquad \dfrac{}{\Gamma \vdash n \rightsquigarrow \mathsf{nat}, \emptyset}$$

$$\dfrac{\Gamma, x : A \vdash t \rightsquigarrow B, \tau}{\Gamma \vdash \lambda x.t \rightsquigarrow A \Rightarrow B, \tau} \qquad \dfrac{\Gamma \vdash t \rightsquigarrow A, \tau \quad \Gamma \vdash u \rightsquigarrow B, \sigma}{\Gamma \vdash tu \rightsquigarrow X, \tau \cup \sigma \cup \{A = B \Rightarrow X\}}\ (X \text{ nueva})$$

$$\frac{\Gamma \vdash t \rightsquigarrow A, \tau \quad \Gamma \vdash u \rightsquigarrow B, \sigma}{\Gamma \vdash t \otimes u \rightsquigarrow \mathsf{nat}, \tau \cup \sigma \cup \{A = \mathsf{nat}, B = \mathsf{nat}\}}$$

$$\frac{\Gamma \vdash t \rightsquigarrow B, \tau \quad \Gamma \vdash u \rightsquigarrow A, \sigma \quad \Gamma \vdash v \rightsquigarrow C, \rho}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v \rightsquigarrow A, \tau \cup \sigma \cup \rho \cup \{B = \mathsf{nat}, A = C\}}$$

$$\frac{\Gamma, x : A \vdash t \rightsquigarrow B, \tau}{\Gamma \vdash \mu x.t \rightsquigarrow B, \tau \cup \{A = B\}} \qquad \frac{\Gamma \vdash u \rightsquigarrow C, \tau \quad \Gamma, x : B \vdash t \rightsquigarrow A, \sigma}{\Gamma \vdash \mathsf{let}\ x = u\ \mathsf{in}\ t \rightsquigarrow A, \tau \cup \sigma \cup \{B = C\}}$$

Ejemplo 5.5.

$$\dfrac{\dfrac{\dfrac{}{f : X \vdash 2 \rightsquigarrow \mathsf{nat}, \emptyset} \quad \dfrac{\dfrac{}{f : X \vdash f \rightsquigarrow X, \emptyset} \quad \dfrac{}{f : X \vdash 1 \rightsquigarrow \mathsf{nat}, \emptyset}}{f : X \vdash f1 \rightsquigarrow Y, \{X = \mathsf{nat} \Rightarrow Y\}}}{f : X \vdash 2 + f1 \rightsquigarrow \mathsf{nat}, \{X = \mathsf{nat} \Rightarrow Y, \mathsf{nat} = \mathsf{nat}, Y = \mathsf{nat}\}}}{\vdash \lambda f.2 + f1 \rightsquigarrow X \Rightarrow \mathsf{nat}, \{X = \mathsf{nat} \Rightarrow Y, \mathsf{nat} = \mathsf{nat}, Y = \mathsf{nat}\}}$$

5.3. Algoritmo de unificación de Robinson

La segunda parte consiste en resolver las ecuaciones sobre los tipos. El lenguaje de tipos no tiene variables y está formado por la constante nat y el símbolo $\Rightarrow$ de dos argumentos. Para resolver las ecuaciones utilizamos el algoritmo de unificación de Robinson, que permite resolver las ecuaciones de cualquier lenguaje sin variables ligadas.

Definición 5.6. Sea θ una substitución $A_1/X_1, \ldots, A_n/X_n$. Decimos que θ es una solución del conjunto de ecuaciones τ si para todo $B = C$ en τ, θB y θC son idénticos.

Teorema 5.7. Si $\vdash t \rightsquigarrow A, \tau$, entonces para toda solución θ de τ, $\vdash t : \theta A$.
Más general: si $\Gamma \vdash t \rightsquigarrow A, \tau$, entonces para toda solución θ de τ, $\theta\Gamma \vdash t : \theta A$, donde $\theta\Gamma$ es la substitución en cada tipo que aparece en Γ.

Algoritmo de unificación de Robinson

- Si una ecuación tiene forma $A \Rightarrow B = C \Rightarrow D$, reemplazarla por las ecuaciones $A = C$ y $B = D$.
- Si una ecuación tiene la forma $\mathsf{nat} = \mathsf{nat}$, borrarla.
- Si una ecuación tiene la forma $\mathsf{nat} = A \Rightarrow B$, o $A \Rightarrow B = \mathsf{nat}$, responder error.
- Si una ecuación tiene forma $X = X$, borrarla.
- Si una ecuación tiene forma $A = X$, $X = A$ y X aparece en A, pero $A \neq X$, responder error.
- Si una ecuación tiene forma $A = X$, $X = A$ y X no aparece en A pero aparece en otras ecuaciones ecuaciones, substituir X por A en todas las ecuaciones.

Si el algoritmo termina en error: en el sistema no había solución.
Si el algoritmo termina sin error, tendremos una lista de ecuaciones $X_1 = A_1, \ldots, X_n = A_n$, con X_i distintas y $\forall ij$, $X_i \notin A_j$. En ese caso, la substitución $\theta = A_1/X_1, \ldots, A_n/X_n$ es una solución.
En ese caso, θ es "principal", en el sentido de que para toda otra solución α, existe una substitución γ tal que $\alpha = \gamma \circ \theta$. Decimos que $\theta = \mathsf{mgu}(\tau)$ ("most general unifier").

Ejemplo 5.8. Continuando con el ejemplo anterior, teníamos

$$\vdash \lambda f.2 + f1 \rightsquigarrow X \Rightarrow \mathsf{nat}, \{X = \mathsf{nat} \Rightarrow Y, \mathsf{nat} = \mathsf{nat}, Y = \mathsf{nat}\}$$

$$\left\{ \begin{array}{c} X = \mathsf{nat} \Rightarrow Y \\ \mathsf{nat} = \mathsf{nat} \\ Y = \mathsf{nat} \end{array} \right. \Rightarrow \left\{ \begin{array}{c} X = \mathsf{nat} \Rightarrow Y \\ Y = \mathsf{nat} \end{array} \right. \Rightarrow \left\{ \begin{array}{c} X = \mathsf{nat} \Rightarrow \mathsf{nat} \\ Y = \mathsf{nat} \end{array} \right.$$

Solución: $[\mathsf{nat} \Rightarrow \mathsf{nat}/X, \mathsf{nat}/Y]$. Por lo tanto,

$$\vdash \lambda f.2 + f1 : (\mathsf{nat} \Rightarrow \mathsf{nat}) \Rightarrow \mathsf{nat}$$

6. Polimorfismo

6.1. Introducción

En la sección anterior vimos que el tipo principal de $\lambda x.x$ es $X \Rightarrow X$. Es decir, $\lambda x.x$ tiene tipo $X \Rightarrow X$ para todo X. Podemos entonces atribuirle el tipo

$$\forall X.(X \Rightarrow X)$$

y agregar una regla según la cual si t tiene tipo $\forall X.A$, entonces t tiene tipo $A[B/X]$ para cualquier B.
A tales lenguajes se los llama polimórficos.

Ejemplo 6.1. Usemos los algoritmos de Hindley y Robinson para inferir el tipo del término $\mathsf{let}\ i = \lambda x.x\ \mathsf{in}\ ii$:

$$\dfrac{\dfrac{\overline{i : X \vdash i \rightsquigarrow X, \emptyset} \quad \overline{i : X \vdash i \rightsquigarrow X, \emptyset}}{i : X \vdash ii \rightsquigarrow Y, \{X = X \Rightarrow Y\}} \quad \dfrac{\overline{x : Z \vdash x \rightsquigarrow Z, \emptyset}}{\vdash \lambda x.x \rightsquigarrow Z \Rightarrow Z, \emptyset}}{\vdash \mathsf{let}\ i = \lambda x.x\ \mathsf{in}\ ii \rightsquigarrow Y, \{X = X \Rightarrow Y, X = Z \Rightarrow Z\}}$$

El algoritmo de Robinson finalizará con error al leer la ecuación $X = X \Rightarrow Y$.

Sin embargo, el término equivalente $(\lambda x.x)(\lambda x.x)$ sí puede ser tipado:

$$\dfrac{\dfrac{\overline{x : X \vdash x \rightsquigarrow X, \emptyset}}{\vdash \lambda x.x \rightsquigarrow X \Rightarrow X, \emptyset} \quad \dfrac{\overline{x : Y \vdash x \rightsquigarrow Y, \emptyset}}{\vdash \lambda x.x \rightsquigarrow Y \Rightarrow Y, \emptyset}}{\vdash (\lambda x.x)(\lambda x.x) \rightsquigarrow Z, \{X \Rightarrow X = (Y \Rightarrow Y) \Rightarrow Z\}}$$

$$X \Rightarrow X = (Y \Rightarrow Y) \Rightarrow Z \implies \left\{ \begin{array}{lcl} X & = & Y \Rightarrow Y \\ X & = & Z \end{array} \right\} \implies Y \Rightarrow Y = Z$$

Es decir, el término tiene tipo $Y \Rightarrow Y$ para cualquier Y.

La diferencia entre el primer término y el segundo es que en el primero, i debe tener el mismo tipo en ambas instancias, $Z \Rightarrow Z$ para cualquier Z, en cambio en el segundo ejemplo, cada función identidad puede instanciar la Z de manera diferente.
Con un tipo $\forall$, podemos hacer que el i del let tenga tipo $\forall X.(X \Rightarrow X)$ y cada ocurrencia de i se instancie en un tipo diferente, haciendo que el término $\mathsf{let}\ i = \lambda x.x\ \mathsf{in}\ ii$ pueda ser tipado como $\forall Y.(Y \Rightarrow Y)$.

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.

6.2. Tipos polimórficos

Podemos dar un tipo cuantificado a las variables ligadas por let, pero vamos a dejar tipos simples para las variables ligadas por λ y fix, de otra manera, está demostrado que no es posible hacer inferencia de tipos.

Tenemos que distinguir tipos sin cuantificadores (los que llamaremos simplemente "tipos") de tipos cuantificados (que llamaremos "esquemas de tipos").

Definición 6.2. Un esquema de tipo tiene forma $\forall X_1 \ldots \forall X_n.A$, donde A es un tipo, con $n \geq 0$.

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

$$A ::= X \mid \mathsf{nat} \mid A \Rightarrow A$$
$$\alpha ::= [A] \mid \forall X.\alpha$$

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

Definición 6.3. Ahora que tenemos variables y un ligador ($\forall$) en los tipos, extendemos la definición de variables libres (FV) para tipos:

$$\begin{aligned}\mathsf{FV}([X]) &= \{X\}\\ \mathsf{FV}([\mathsf{nat}]) &= \emptyset\\ \mathsf{FV}([A \Rightarrow B]) &= \mathsf{FV}([A]) \cup \mathsf{FV}([B])\\ \mathsf{FV}(\forall X.\alpha) &= \mathsf{FV}(\alpha) \setminus \{X\}\end{aligned}$$

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

Definición 6.4. El sistema de tipos asocia contextos y términos con esquemas de tipos, $\Gamma \vdash t : \alpha$, y viene dado por

$$\frac{}{\Gamma, x : \alpha \vdash x : \alpha}\ ax_v \qquad \frac{}{\Gamma \vdash n : [\mathsf{nat}]}\ ax_c$$

$$\frac{\Gamma, x : [A] \vdash t : [B]}{\Gamma \vdash \lambda x.t : [A \Rightarrow B]}\ \Rightarrow_i \qquad \frac{\Gamma \vdash t : [A \Rightarrow B] \quad \Gamma \vdash u : [A]}{\Gamma \vdash tu : [B]}\ \Rightarrow_e$$

$$\frac{\Gamma \vdash t : [\mathsf{nat}] \quad \Gamma \vdash u : [\mathsf{nat}]}{\Gamma \vdash t \otimes u : [\mathsf{nat}]}\ \otimes \qquad \frac{\Gamma \vdash t : [\mathsf{nat}] \quad \Gamma \vdash u : [A] \quad \Gamma \vdash v : [A]}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v : [A]}\ \mathsf{ifz}$$

$$\frac{\Gamma, x : [A] \vdash t : [A]}{\Gamma \vdash \mu x.t : [A]}\ \mathsf{fix} \qquad \frac{\Gamma \vdash t : \alpha \quad \Gamma, x : \alpha \vdash u : [A]}{\Gamma \vdash \mathsf{let}\ x = t\ \mathsf{in}\ u : [A]}\ \mathsf{let}$$

$$\text{Si } X \notin \mathsf{FV}(\Gamma),\ \frac{\Gamma \vdash t : \alpha}{\Gamma \vdash t : \forall X.\alpha}\ \forall_i \qquad \frac{\Gamma \vdash t : \forall X.\alpha}{\Gamma \vdash t : \alpha[A/X]}\ \forall_e$$

Observaciones.

- En la definición anterior se atribuye un esquema de tipo a cada término, en particular a las variables. Las reglas fix y λ piden $x : [A]$, es decir, un esquema sin cuantificar. Sólo let permite darle a la variable cualquier esquema.
- La condición de la regla $\forall_i$ permite tipar, por ejemplo,

$$\dfrac{\dfrac{\dfrac{\dfrac{}{x : [X] \vdash x : [X]}\, ax_v}{\vdash \lambda x.x : [X \Rightarrow X]}\, \Rightarrow_i}{\vdash \lambda x.x : \forall X.[X \Rightarrow X]}\, \forall_i}{\vdash \lambda x.x : [\mathsf{nat} \Rightarrow \mathsf{nat}]}\, \forall_e$$

  pero impide algo como

$$\dfrac{\dfrac{}{x : [X] \vdash x : [X]}\, ax_v}{x : [X] \vdash x : \forall X.[X]}\, \forall_i$$

Ejercicios: Tipar los siguientes términos con tipos polimórficos

1. $\lambda x.x$
2. let $i = \lambda x.x$ in ii
3. $(\lambda f.ff)(\lambda x.x)$
4. $(\lambda x.xx)(\lambda x.xx)$

Teorema 6.5. Todo término tipado que no contenga a fix, termina.

6.3. 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).
Primero definimos la siguiente relación entre tipos.

$$\alpha \preceq \forall X.\alpha, \text{ si } X \notin \mathsf{FV}(\alpha) \qquad \text{y} \qquad \forall X.\alpha \preceq \alpha[A/X]$$

El sistema es el siguiente:

$$\frac{\alpha \preceq \beta}{\Gamma, x : \alpha \vdash x : \beta} \qquad \frac{}{\Gamma \vdash n : [\mathsf{nat}]} \qquad \frac{\Gamma, x : [A] \vdash t : [B]}{\Gamma \vdash \lambda x.t : [A \Rightarrow B]} \qquad \frac{\Gamma \vdash t : [A \Rightarrow B] \quad \Gamma \vdash u : [A]}{\Gamma \vdash tu : [B]}$$

$$\frac{\Gamma \vdash t : [\mathsf{nat}] \quad \Gamma \vdash u : [\mathsf{nat}]}{\Gamma \vdash t \otimes u : [\mathsf{nat}]} \qquad \frac{\Gamma \vdash t : [\mathsf{nat}] \quad \Gamma \vdash u : [A] \quad \Gamma \vdash v : [A]}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v : [A]}$$

$$\frac{\Gamma, x : [A] \vdash t : [A]}{\Gamma \vdash \mu x.t : [A]} \qquad \frac{\Gamma \vdash u : [A] \quad \Gamma, x : \vec{\Gamma}(A) \vdash t : [B]}{\Gamma \vdash \mathsf{let}\ x = u\ \mathsf{in}\ t : [B]} \quad \begin{array}{l} \vec{\Gamma}(A) = \forall \vec{X}.[A], \\ \vec{X} = \mathsf{FV}(A) \setminus \mathsf{FV}(\Gamma) \end{array}$$

Observación. Si permitimos $\forall$ en λ obtenemos "System F" (de Girard y Reynolds), pero la tipabilidad es indecidible (teorema de Wells), es decir, no existe algoritmo de inferencia. Similarmente, $\forall$ en fix hace el sistema indecidible (teorema de Kfoury).
Por eso, dar polimorfismo sólo a let es un buen compromiso que permite reusabilidad de código e inferencia de tipos.

6.4. Sistema F

Los tipos en Sistema F son los mismos de tipos simples (ver Definición 4.1) , a los que se agregan variables y para-todo ligando esas variables:

$$A ::= \mathsf{nat} \mid A \Rightarrow A \mid X \mid \forall X.A$$

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 4.2), a las que se agregan las reglas de introducción y eliminación del para-todo.

$$\frac{}{\Gamma, x : A \vdash x : A}\ ax_v \qquad \frac{}{\Gamma \vdash n : \mathsf{nat}}\ ax_c$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \Rightarrow B}\ \Rightarrow_i \qquad \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash tu : B}\ \Rightarrow_e$$

$$\frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash u : \mathsf{nat}}{\Gamma \vdash t \otimes u : \mathsf{nat}}\ \otimes \qquad \frac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash u : A \quad \Gamma \vdash v : A}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v : A}\ \mathsf{ifz}$$

$$\frac{\Gamma, x : A \vdash t : A}{\Gamma \vdash \mu x : A.t : A}\ \mathsf{fix} \qquad \frac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{let}\ x : A = u\ \mathsf{in}\ t : B}\ \mathsf{let}$$

$$\frac{\Gamma \vdash t : A \quad X \notin \mathsf{FV}(\Gamma)}{\Gamma \vdash t : \forall X.A}\ \forall_i \qquad \frac{\Gamma \vdash t : \forall X.A}{\Gamma \vdash t : A[B/X]}\ \forall_e$$

7. Formas de llevar a cabo la ejecución

7.1. Interpretación

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

7.1.1. Interpretación en CBN

Supongamos que queremos un programa 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 7.1. 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 $x = t$, donde t no es un valor, deberíamos encontrar también el contexto en el que t debe ser evaluado.

Definición 7.2 (Thunk)**.** Un par $\langle t, \Gamma \rangle$ formado por un término y un contexto de evaluación, se llama thunk.

Definición 7.3 (Cierre)**.** Cuando queremos en cambio evaluar un término $\lambda x.t$ en un contexto, el resultado no puede ser simplemente $\lambda x.t$, tiene que ser t pero con t "cerrado". Introducimos para esto otro valor llamado cierre que se compone de un término $\lambda x.t$ y un contexto Γ, y se nota $\langle x, t, \Gamma \rangle$.

Definición 7.4 (Relación $\hookrightarrow$ en CBN)**.** Vamos a definir la relación $\Gamma \vdash t \hookrightarrow v$ que se lee "t se interpreta a v en Γ".

$$\frac{\Gamma' \vdash t \hookrightarrow v}{\Gamma, x = \langle t, \Gamma' \rangle, \Delta \vdash x \hookrightarrow v}\ x \notin \mathsf{D}(\Delta) \qquad \frac{}{\Gamma \vdash n \hookrightarrow n} \qquad \frac{\Gamma \vdash t \hookrightarrow n \quad \Gamma \vdash u \hookrightarrow m}{\Gamma \vdash t \otimes u \hookrightarrow p}\ \text{Si } n \otimes m = p$$

$$\frac{}{\Gamma \vdash \lambda x.t \hookrightarrow \langle x, t, \Gamma \rangle} \qquad \frac{\Gamma \vdash t \hookrightarrow \langle x, t', \Gamma' \rangle \quad \Gamma', x = \langle r, \Gamma \rangle \vdash t' \hookrightarrow v}{\Gamma \vdash tr \hookrightarrow v}$$

$$\frac{\Gamma \vdash t \hookrightarrow 0 \quad \Gamma \vdash r \hookrightarrow v}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s \hookrightarrow v} \qquad \frac{\Gamma \vdash t \hookrightarrow n \quad \Gamma \vdash s \hookrightarrow v}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s \hookrightarrow v}\ \text{Si } n \neq 0$$

$$\frac{\Gamma, x = \langle r, \Gamma \rangle \vdash t \hookrightarrow v}{\Gamma \vdash \mathsf{let}\ x = r\ \mathsf{in}\ t \hookrightarrow v} \qquad \frac{\Gamma, x = \langle \mu x.t, \Gamma \rangle \vdash t \hookrightarrow v}{\Gamma \vdash \mu x.t \hookrightarrow v}$$

Ejemplo 7.5. (Leer de abajo hacia arriba)

$$\dfrac{\dfrac{}{\vdash \lambda x.x \times x \hookrightarrow \langle x, x \times x, \emptyset \rangle} \qquad \dfrac{\dfrac{\dfrac{}{\vdash 4 \hookrightarrow 4}}{x = \langle 4, \emptyset \rangle \vdash x \hookrightarrow 4} \quad \dfrac{\dfrac{}{\vdash 4 \hookrightarrow 4}}{x = \langle 4, \emptyset \rangle \vdash x \hookrightarrow 4}}{x = \langle 4, \emptyset \rangle \vdash x \times x \hookrightarrow 16}}{\vdash (\lambda x.x \times x)4 \hookrightarrow 16}$$

7.1.2. Interpretación en CBV

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

$$\frac{\Gamma' \vdash t \hookrightarrow v}{\Gamma, x = \langle t, \Gamma' \rangle, \Delta \vdash x \hookrightarrow v}\ x \notin \mathsf{D}(\Delta) \quad \text{tenemos} \quad \frac{}{\Gamma, x = v, \Delta \vdash x \hookrightarrow v}\ x \notin \mathsf{D}(\Delta)$$

Lo mismo sucede con let. Así que los contextos ligan variables con valores, no con thunks. Sin embargo, la regla fix 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 $\mu x.t$ y un contexto.

Definición 7.6 (Relación $\hookrightarrow$ en CBV)**.**

$$\frac{}{\Gamma, x = v, \Delta \vdash x \hookrightarrow v}\ x \notin \mathsf{D}(\Delta) \qquad \frac{\Gamma' \vdash \mu y.t \to v}{\Gamma, x = \langle \mu y.t, \Gamma \rangle', \Delta \vdash x \hookrightarrow v}\ x \notin \mathsf{D}(\Delta)$$

$$\frac{}{\Gamma \vdash n \hookrightarrow n} \qquad \frac{\Gamma \vdash t \hookrightarrow n \quad \Gamma \vdash u \hookrightarrow m}{\Gamma \vdash t \odot u \hookrightarrow p}\ \text{Si } n \odot m = p$$

$$\frac{}{\Gamma \vdash \lambda x.t \hookrightarrow \langle x, t, \Gamma\rangle} \qquad \frac{\Gamma \vdash r \hookrightarrow w \quad \Gamma \vdash t \hookrightarrow \langle x, t', \Gamma'\rangle \quad \Gamma', x = w \vdash t' \hookrightarrow v}{\Gamma \vdash tr \hookrightarrow v}$$

$$\frac{\Gamma \vdash t \hookrightarrow 0 \quad \Gamma \vdash r \hookrightarrow v}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s \hookrightarrow v} \qquad \frac{\Gamma \vdash t \hookrightarrow n \quad \Gamma \vdash s \hookrightarrow v}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s \hookrightarrow v}\ \text{Si } n \neq 0$$

$$\frac{\Gamma \vdash r \hookrightarrow w \quad \Gamma, x = w \vdash t \hookrightarrow v}{\Gamma \vdash \mathsf{let}\ x = r\ \mathsf{in}\ t \hookrightarrow v} \qquad \frac{\Gamma, x = \langle \mu x.t, \Gamma\rangle \vdash t \hookrightarrow v}{\Gamma \vdash \mu x.t \hookrightarrow v}$$

7.2. Compilación

¿Quién interpreta al intérprete? ¡Hay que escribirlo en lenguaje máquina!
Alternativa: olvidar los inérpretes y escribir un compilador.

Un intérprete es un programa que toma un término de PCF y devuelve su valor.
Un compilador es un programa que toma un término de PCF y devuelve un programa en lenguaje máquina tal que al ejecutarlo devuelve su valor.

Ventajas.

- Se traduce una sóla vez, no a cada ejecución.
- Una vez compilado, la ejecución es muy rápida.
- ¡Un compilador puede compilarse a sí mismo! (un intérprete, no)

Máquina virtual La idea es compilar a una máquina abstracta, luego esta máquina es ejecutada en diferentes sistemas y computadoras. Así, el programa original puede ser compilado una sóla vez hacia esa máquina virtual, y distribuir el programa ya compilado, que funcionará en cualquier entorno que pueda correr la máquina virtual.

Ejemplo 7.7 (de Compilador). Supongamos que la máquina tiene las siguientes instrucciones:

Ldi n	Poner el entero n en el acumulador
Push	Poner el contenido del acumulador en la pila
Add	Sumar el primero de la pila con el acumulador, poner el resultado en el acumulador y borrar el elemento de la pila.

Entonces, el término

$$((((1 + 2) + 3) + 4) + 5) + 6$$

compila a

Ldi 6; Push; Ldi 5; Push; Ldi 4; Push; Ldi 3; Push; Ldi 2; Push; Ldi 1; Add; Add; Add; Add; Add;

7.2.1. Una máquina abstracta para PCF

En el ejemplo anterior vimos cómo compilaríamos un lenguaje que sólo tenga números y suma, lo cual es un (pequeño) fragmento de PCF. Ahora queremos generalizar eso a todo PCF.
En PCF un término necesita un entorno para interpretarse, así que además de la pila, el acumulador y el código, tendremos que tener también un entorno.

Para compilar, vamos a asegurarnos de que el fix sólo se aplique a funciones, y por lo tanto, vamos a reemplazar $\mu f.\lambda x.t$ por fixfun $f\ x.t$.
La máquina abstracta tiene las siguientes instrucciones:

Ldi n	Poner el entero n en el acumulador
Push	Poner el contenido del acumulador en la pila
Add	Sumar el primero de la pila con el acumulador, poner el resultado en el acumulador y borrar el elemento de la pila
Sub	Restar el primero de la pila con el acumulador, poner el resultado en el acumulador y borrar el elemento de la pila
Mult	Multiplicar el primero de la pila con el acumulador, poner el resultado en el acumulador y borrar el elemento de la pila
Div	Dividir el primero de la pila con el acumulador, poner el resultado en el acumulador y borrar el elemento de la pila
Extend	Extender el entorno agregando el contenido del acumulador en la última posición
Search n	Buscar el valor en la posición n en el entorno y ponerlo en el acumulador
Pushenv	Meter el contenido del entorno arriba de la pila
Popenv	Sacar lo de arriba de la pila a un entorno
Mkclos(i)	Meter el cierre $\langle i, e\rangle$ en el acumulador, donde e es el entorno actual
Apply	Toma un cierre $\langle i, e\rangle$ del acumulador y pone el entorno $e, \langle i, e\rangle, W$, donde W es el elemento de arriba de la pila; retira el elemento de arriba de la pila, y pone el resto de las instrucciones i en el registro de código
Test(i, j)	Meter i o j en el acumulador dependiendo de si el acumulador es 0 o $n > 0$

Semántica operacional de la máquina abstacta El estado de la máquina, el contenido de sus registros, es una cuadrupla formada por un valor (el acumulador), una lista donde los elementos son o bien valores o bien listas de valores (la pila), una lista de variables y valores (el entorno), y una secuencia de instrucciones (el código). Utilizamos notación usual para listas y ";" para secuencias de código.
La semántica operacional viene dada por

$$
\begin{aligned}
(a, s, e, \mathsf{Mkclos}\ i; c) &\to (\langle i, e\rangle, s, e, c) \\
(a, s, e, \mathsf{Push}; c) &\to (a, a{:}s, e, c) \\
(a, s, e, \mathsf{Extend}; c) &\to (a, s, e +\!\!+ [a], c) \\
(a, s, e, \mathsf{Search}\ n; c) &\to (v, s, e, c) && \text{Si } v \text{ es el } n\text{-ésimo valor en } e \text{ comenzando desde la derecha} \\
(a, s, e, \mathsf{Pushenv}; c) &\to (a, e{:}s, e, c) \\
(a, e'{:}s, e, \mathsf{Popenv}; c) &\to (a, s, e', c) \\
(\langle i, e'\rangle, w{:}s, e, \mathsf{Apply}; c) &\to (\langle i, e'\rangle, s, e' +\!\!+ [\langle i, e'\rangle, w], i; c) \\
(a, s, e, \mathsf{Ldi}\ n; c) &\to (n, s, e, c) \\
(n, m{:}s, e, \mathsf{Add}; c) &\to (n + m, s, e, c)
\end{aligned}
$$

$$
\begin{aligned}
(n, m; s, e, \mathsf{Sub}; c) &\to (n - m, s, e, c) \\
(n, m; s, e, \mathsf{Mult}; c) &\to (n * m, s, e, c) \\
(n, m; s, e, \mathsf{Div}; c) &\to (n/m, s, e, c) \\
(0, s, e, \mathsf{Test}(i, j); c) &\to (0, s, e, i; c) \\
(n, s, e, \mathsf{Test}(i, j); c) &\to (n, s, e, j; c) \qquad \text{Si } n > 0
\end{aligned}
$$

Un término irreducible es una cuadrupla en el que la cuarta componente (el código) está vacío. Si i es una secuencia de instrucciones y si el término $(0, [], [], i)$ reduce a un término irreducible de la forma $(v, [], [], \_)$, decimos que v es el resultado de ejecutar i y lo notamos $i \Rightarrow v$.

Compilando PCF La compilación se hará reemplazando un término t por una secuencia de instrucciones, notadas $|t|$, siguiendo las siguientes reglas.

$$
\begin{aligned}
|x|_e &= \mathsf{Search}\ n \quad \text{donde } n \text{ es la posición de } x \text{ en } e \text{ desde la derecha} \\
|tu|_e &= \mathsf{Pushenv}; |u|_e; \mathsf{Push}; |t|_e; \mathsf{Apply}; \mathsf{Popenv} \\
|\lambda x.t|_e &= \mathsf{Mkclos}\ |t|_{e+[\_,x]} \\
|\mathsf{fixfun}\ f\ x.t|_e &= \mathsf{Mkclos}\ |t|_{e+[f,x]} \\
|n|_e &= \mathsf{Ldi}\ n \\
|t + u|_e &= |u|_e; \mathsf{Push}; |t|_e; \mathsf{Add} \\
|t - u|_e &= |u|_e; \mathsf{Push}; |t|_e; \mathsf{Sub} \\
|t * u|_e &= |u|_e; \mathsf{Push}; |t|_e; \mathsf{Mult} \\
|t/u|_e &= |u|_e; \mathsf{Push}; |t|_e; \mathsf{Div} \\
|\mathsf{ifz}\ t\ \mathsf{then}\ u\ \mathsf{else}\ v|_e &= |t|_e; \mathsf{Test}(|u|_e; |v|_e) \\
|\mathsf{let}\ x = t\ \mathsf{in}\ u|_e &= \mathsf{Pushenv}; |t|_e; \mathsf{Extend}; |u|_{e+[x]}; \mathsf{Popenv}
\end{aligned}
$$

Ejemplo 7.8. El término

$$\mathsf{let}\ f = \mathsf{fixfun}\ f\ x.\mathsf{ifz}\ x\ \mathsf{then}\ 1\ \mathsf{else}\ (x * (f(x - 1)))\ \mathsf{in}\ f6$$

compila a la siguiente secuencia de instrucciones:

$$
\begin{aligned}
&\mathsf{Pushenv}; \\
&\mathsf{Mkclos}\ (\mathsf{Search}\ 0; \mathsf{Test}(\mathsf{Ldi}\ 1; \\
&(\mathsf{Pushenv}; \mathsf{Ldi}\ 1; \mathsf{Push}; \mathsf{Search}\ 0; \mathsf{Sub}; \mathsf{Push}; \mathsf{Search}\ 1; \mathsf{Apply}; \mathsf{Popenv}; \mathsf{Push}; \mathsf{Search}\ 0; \mathsf{Mult}))); \\
&\mathsf{Extend}; \\
&\mathsf{Pushenv}; \mathsf{Ldi}\ 6; \mathsf{Push}; \mathsf{Search}\ 0; \mathsf{Apply}; \mathsf{Popenv}; \\
&\mathsf{Popenv}
\end{aligned}
$$

y el resultado de la ejecución es 720.

Teorema 7.9. Si v es un valor entero, $t \hookrightarrow v$ si y sólo si $|t| \Rightarrow v$.

8. Registros y Objetos

8.1. Registros

8.1.1. Campos etiquetados

Supongamos que queremos una función que tome la edad, altura y peso de una persona, y calcule algo, como el índice de masa corporal. Podríamos hacer una función que reciba tres parámetros, en órden, donde el primero sea la edad, el segundo la altura y el tercero el peso. O podríamos extender el lenguaje con tuplas y hacer que la función reciba una tupla donde el primer elemento es la edad, y el segundo es una tupla cuyo primer elemento es la altura y el segundo el peso. O podríamos usar 3-tuplas, o n-tuplas, es decir listas, y directamente pasar (edad, altura, peso). Sin embargo, en todas estas soluciones, la forma de referenciar a la edad, altura o peso es saber su órden en la estructura. Es decir que podemos hacer una función:

$$f(x) := \begin{cases} \text{edad} & \text{Si } x = 0 \\ \text{altura} & \text{Si } x = 1 \\ \text{peso} & \text{Si } x = 2 \end{cases} \qquad \text{donde edad, altura, peso} \in \mathbb{R}$$

Es decir, que esta es una función de $\{0, 1, 2\}$ en reales, donde 0 se asocia a la edad, 1 a la altura y 2 al peso. La elección es totalmente arbitraria. Lo lógico sería asociar la palabra "edad" al número que representa la edad, "altura" al número que representa la altura y "peso" al número que representa el peso. Más aún, podemos asociar "edad" a un número natural del conjunto $\{0, \ldots, 150\}$, "altura" a un número real del intervalo $[0.3; 3.5]$, y peso a un número real del intervalo $[1; 300]$.

8.1.2. Extensión de PCF con registros

Gramática Sea L in conjunto infinito de etiquetas válidas. Entonces, extendemos la gramática de PCF de la siguiente manera:

$$\begin{aligned} t ::= x \mid \lambda x.t \mid tt \mid n \in \mathbb{N} \mid t + t \mid t - t \mid t \times t \mid t/t \\ \mid \mathsf{ifz}\ t\ \mathsf{then}\ t\ \mathsf{else}\ t \mid \mu x.t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \\ \mid \{l_1 = t, l_2 = t, \ldots, l_n = t\} \mid t \cdot l \mid t(l \leftarrow t) \end{aligned}$$

donde $l, l_1, \ldots, l_n \in L$.
Ninguno de los tres nuevos símbolos liga variables.

- El símbolo $\{\}$ es un constructor con $2n$ argumentos, donde los argumentos impares son etiquetas y los argumentos pares son términos. Este símbolo permite construir un registro.
- El símbolo $\cdot$ es un constructor con 2 argumentos, donde el primero es un término y el segundo una etiqueta, y permite acceder a un registro.
- El símbolo $\leftarrow$ es un constructor con 3 argumentos donde el primero es un término, el segundo una etiqueta y el tercero un término. Permite construir un nuevo registro idéntico a uno ya construido, excepto en un único campo.

Semántica operacional Extendemos la semántica operacional de PCF con las siguientes reglas:

$$\{l_1 = t_1, \ldots, l_n = t_n\} \cdot l_i \to t_i$$
$$\{l_1 = t_1, \ldots, l_n = t_n\}(l_i \leftarrow u) \to \{l_1 = t_1, \ldots, l_{i-1} = t_{i-1}, l_i = u, l_{i+1} = t_{i+1}, \ldots, l_n = t_n\}$$

En call-by-value vamos a reducir siempre los términos dentro del registro antes de referenciarlos con $\cdot$, mientras que en call-by-name, no.
Y acá tenemos los mismos problemas y ventajas que con las funciones en call-by-value y call-by-name. Por ejemplo, $\mathsf{let}\ x = \{a = \mathsf{fact}\ 10, b = 4\}\ \mathsf{in}\ x \cdot b$ en call-by-value calcula el factorial de 10, que será descartado, mientras en call-by-name no. Por otro lado, $\mathsf{let}\ x = \{a = \mathsf{fact}\ 10, b = 4\}\ \mathsf{in}\ x \cdot a + x \cdot a$ calcula dos veces el factorial en call-by-name, mientras lo hace sólo una vez en call-by-value. La interpretación de $\mathsf{let}\ x = \{a = \mu y.y, b = 4\}\ \mathsf{in}\ x \cdot b$ loopea en call-by-value y devuelve 4 en call-by-name.

8.2. Objetos

Un programa trata en general datos muy diversos, muchas veces expresados por registros. Por ejemplo, el sistema informático de una empresa trata los pedidos de pago, los recibos de sueldo, etc, y un pedido de pago es un registro que contiene la identificación de el objeto comprado, la cantidad comprada, etc. Hay muchas posibilidades para imprimir los datos. Podemos escribir una función imprimir única que comience por testear la naturaleza de los datos (pedido de pago, recibo de sueldo, etc), y que imprima según un cierto formato en función de su naturaleza. O podemos escribir muchas funciones, imprimir_pedido_de_pago, imprimir_recibo_de_sueldo ,etc. Y podríamos tener un registro llamado imprimir donde cada uno de sus campos sea una función de impresión. Otra solución es que la función de impresión para un registro dado sea parte del registro. A esta clase de datos se les llama clase, y a sus elementos, objetos.
En la forma más extrema de programación con objetos, cada objeto, por ejemplo cada pedido de pago, es un registro que contiene una función imprimir diferente. Un pedido de pago entonces es un registro que contiene, además de los campos habituales (la identificación del objeto comprado, la cantidad, el precio), un campo imprimir que es una función de impresión que podemos aplicar al objeto en sí mismo para imprimirlo.
Algunos lenguajes, como Java, asocian una función imprimir, no a cada objeto, sino a cada clase de objetos. Así, dos objetos de la misma clase comparten una misma función de impresión. Si queremos que dos objetos t y u de la misma clase C no compartan la misma función de impresión, será necesario definir dos subclases T y U de C que hereden los campos de C pero redefinan el campo imprimir.

8.2.1. Los métodos y los campos funcionales

Un objeto es un registro donde ciertos campos son funciones. En un lenguaje como Java, donde las funciones no son elementos de primera clase, estamos obligados a distinguir los campos no funcionales de campos funcionales, a los cuales llamamos métodos del objeto. En un lenguaje donde las funciones son elementos de primera clase, como PCF, eso no es necesario. Los objetos son registros como los otros y entonces ya podemos empezar a programar con objetos utilizando la extensión de PCF que dimos en la sección anterior.

Ejercicio Un programa de gestión de la boletería de un concierto es un objeto con los siguientes campos:

- Un entero que representa el número de lugares disponibles en campo.
- Un entero que representa el número de lugares disponibles en la platea.
- Una función que toma como argumento un objeto y un entero (0 para el campo, 1 para la platea) y que devuelve 0 o 1 según la reserva esté cerrada o que aún queden lugares en esta categoría.
- Una función que toma como argumento un objeto y un entero (0 para el campo, 1 para la platea) y que reserva un lugar decrementando en una unidad el número de lugares disponibles en esa categoría.

Programar este objeto en PCF con registros.

8.2.2. ¿Qué es "self"?

Si t es el objeto construido en el ejercicio de la sección anterior, para saber si la reservación está cerrada para el campo o si quedan lugares, interpretamos el término $t \cdot \mathsf{libre}\ t0$. En efecto, la función $t \cdot \mathsf{libre}$ toma como argumento un objeto u cualquiera y un entero n que indica si el campo de u correspondiente a n (campo si $n = 0$, platea si $n = 1$) es nulo o no.
Ahora queremos que el método libre del objeto t se aplique al objeto t mismo, es decir, que no lo invoquemos con $t \cdot \mathsf{libre}\ t0$, sino simplemente $t\#\mathsf{libre}\ 0$.
Una manera de hacerlo es considerar el término $t\#l$ como una notación para $t \cdot l\ t$. El problema de eso es que si t es un objeto y l es una etiqueta de t, no podemos utilizar $t\#l$ al menos que el campo l sea una función de tipo $A \Rightarrow \ldots$, donde A es el tipo de t mismo. Dicho de otra manera, no podemos utilizar $t\#l$ salvo que l sea la etiqueta de un método. Si l es la etiqueta de un campo que no es un método, sólo podemos utilizar $t \cdot l$.
Podemos fijar esta diferencia entre los métodos y los otros campos imponiendo que todos los campos de un objeto sean funciones. Un campo de un objeto t cuyo valor es el entero 3 se transforma en un campo cuyo valor es la función $\lambda s.3$. Así, el término $t\#a = t \cdot a\ t = (\lambda s.3)t \to 3$.
Llamamos self o this a la variable ligada al primer argumento de todos los métodos de un objeto. En efecto, en la mayoría de los lenguajes de programación, se utiliza una variable self o this implícitamente ligada a la construcción del objeto, que sirve para designar el objeto en sí mismo.

8.2.3. PCF con objetos

Para forzar que todos los campos de un objeto sean funciones, podemos modificar ligeramente PCF con registros, para tener PCF con objetos propiamente dichos.

$$\begin{aligned} t ::= {} & x \mid \lambda x.t \mid tt \mid n \in \mathbb{N} \mid t + t \mid t - t \mid t \times t \mid t/t \\ & \mid \mathsf{ifz}\ t\ \mathsf{then}\ t\ \mathsf{else}\ t \mid \mu x.t \mid \mathsf{let}\ x = t\ \mathsf{in}\ t \\ & \mid \{l_1 = \sigma s.t, l_2 = \sigma s.t, \ldots, l_n = \sigma s.t\} \mid t\#l \mid t(l \leftarrow \sigma s.t) \end{aligned}$$

donde $l, l_1, \ldots, l_n \in L$, $s \in \mathsf{Vars}$.
La semántica operacional también debe actualizarse:

$$\{l_1 = \sigma s.t_1, \ldots, l_n = \sigma s.t_n\} \# l_i \to t_i[\{l_1 = \sigma s.t_1, \ldots, l_n = \sigma s.t_n\}/s_i]$$

$$\{l_1 = \sigma s.t_1, \ldots, l_n = \sigma s.t_n\}(l_i \leftarrow \sigma s.u_i)$$
$$\to \{l_1 = \sigma s.t_1, \ldots, l_{i-1} = \sigma s.t_{i-1}, l_i = \sigma s.u_i, l_{i+1} = \sigma s.t_{i+1}, \ldots, l_n = \sigma s.t_n\}$$

9. Semántica denotacional

9.1. Primeras definiciones

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

Semántica: Qué significan.

Ejemplo: "A perro un". Sintacticamente correcto. Semánticamente incorrecto, ya que la frase no significa nada.

Definición 9.1 (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 $[\![p]\!]$. (Lo veremos más adelante en esta sección).

Semántica operacional a grandes pasos También llamada semántica operacional a grandes pasos o semántica natural. Consiste en dar una definición inductiva de $\hookrightarrow$ que nos relacione un término con su valor. Por ejemplo, el intérprete de la Sección 7.1:

$$\underbrace{(\lambda x.\lambda y.x + y)((\lambda x.x)4)5}_{\text{Significado de esta expresión: } 9} \hookrightarrow 9$$

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

$$(\lambda x.\lambda y.x + y)4\ 5 \quad \text{y} \quad (\lambda x.\lambda y.x + y)5\ 4$$

puedo ver que los 3 programas tienen la misma semántica.

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:

$$(\lambda x.(x \times x) + x)4 \to (4 \times 4) + 4 \to 16 + 4 \to 20$$

$$t \hookrightarrow r \iff t \to^* r \qquad \text{y } r \text{ irreducible}$$

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

La no terminación. 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 t no termina, entonces no existe r tal que $t \hookrightarrow r$.
La segunda consiste en agregar un elemento particular $\perp$ a los valores de salida y considerar que si t no termina, entonces $t \hookrightarrow \perp$.

9.2. La semántica denotacional de PCF tipado

En general, en los lenguajes funcionales buscamos reducir la distancia que separa la noción de programa de la de función. Es decir, se busca reducir la distancia entre un programa y su semántica denotacional.

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

$$\begin{aligned} [\![\mathsf{nat}]\!] &= \mathbb{N} \\ [\![A \Rightarrow B]\!] &= [\![A]\!] \to [\![B]\!] \end{aligned}$$

donde $A \to B$ es el conjunto de funciones de A en B.

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

$$\begin{aligned} [\![x]\!]_\theta &= \theta(x) \\ [\![\lambda x : A.t]\!]_\theta &= a^{\in [\![A]\!]} \mapsto [\![t]\!]_{\theta, x=a} \\ [\![tr]\!]_\theta &= [\![t]\!]_\theta [\![r]\!]_\theta \\ [\![n]\!]_\theta &= n \\ [\![t \otimes r]\!]_\theta &= [\![t]\!]_\theta \otimes [\![r]\!]_\theta \\ [\![\mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s]\!]_\theta &= \begin{cases} [\![r]\!]_\theta & \text{si } [\![t]\!]_\theta = 0 \\ [\![s]\!]_\theta & \text{si } [\![t]\!]_\theta \in \mathbb{N}^* \end{cases} \\ [\![\mathsf{let}\ x : A = r\ \mathsf{in}\ t]\!]_\theta &= [\![t]\!]_{\theta, x=[\![r]\!]_\theta} \end{aligned}$$

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. $t/0$ tirar error en PCF y no está definida en matemática. Para que esta definición sea correcta hay que agregar un elemento error a cada conjunto $[\![A]\!]$ y adaptar la definición
$$[\![t/r]\!] = \begin{cases} \mathsf{error} & \text{si } [\![r]\!] = 0 \\ [\![t]\!]/[\![r]\!] & \text{si } [\![t]\!] \in \mathbb{N}, [\![r]\!] \in \mathbb{N}^* \end{cases}$$
2. Aún no dijimos como interpretar fix.

La construcción fix 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 PCF, sólo podemos tomar un punto fijo si este existe y si hay varios, ¡tenemos que especificar cual!

Ejemplo 9.2.

1. $f(x) = x + 1$ no tiene punto fijo. Pero en PCF $\mu x : \mathsf{nat}.x + 1$ es válido.
2. $\lambda f : \mathbb{N} \to \mathbb{N}.\lambda x : \mathbb{N}.(fx) + 1$ tampoco tiene punto fijo... y basta con camabiar el primer λ y tengo el fix.
3. $\lambda x : \mathbb{N}.x$ tiene infinitos puntos fijos.

Teorema 9.3. Si tomamos el punto fijo de una función que no tiene punto fijo o que tiene varios, el programa que obtenemos no termina.

Observación. También existen programas con un sólo punto fijo y que no terminan en PCF. Por ejemplo $\lambda x : \mathsf{nat}.x + x$.

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 $\mu x : \mathsf{nat}.(x + 1) \hookrightarrow \bot$.
- En la semántica denotacional la idea es hacer lo mismo: definir una función parcial $[\![\cdot]\!]$ tal que $[\![\mu x : \mathsf{nat}.(x + 1)]\!]$ no esté definido, y adjuntamos un valor $\bot$ a $[\![\mathsf{nat}]\!]$ tal que

$$[\![\mu x : \mathsf{nat}.(x + 1)]\!] = \bot$$

Agregando el valor $\bot$, cuando interpretamos, por ejemplo $t + r$, comenzamos por interpretar t y r y si alguno loopea, también lo hace $t + r$. Entonces:

$$[\![t + r]\!]_\theta = \begin{cases} [\![t]\!]_\theta + [\![r]\!]_\theta & \text{si } [\![t]\!]_\theta, [\![r]\!]_\theta \in \mathbb{N} \\ \bot & \text{si } [\![t]\!]_\theta = \bot \text{ o } [\![r]\!]_\theta = \bot \end{cases}$$

Ahora vemos que la función $[\![\lambda x : \mathsf{nat}.x + 1]\!]$ que no tenía punto fijo cuando $[\![\mathsf{nat}]\!] = \mathbb{N}$, si $[\![\mathsf{nat}]\!] = \mathbb{N} \cup \bot$ le podemos dar $\bot$ como semántica a ese término.
$[\![\lambda x : \mathsf{nat}.x]\!]$ que tenía muchos puntos fijos, ahora tiene uno más: $\bot$.
$[\![\lambda x : \mathsf{nat}.x + 1]\!]$ 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 9.4 (Orden de Scott)**.** $\bot$ es el elemento más chico de cualquier conjunto que lo contenga.

Definimos entonces $[\![\mu x : A.t]\!]$ como el punto fijo más chico de $[\![\lambda x : A.t]\!]$.

Semántica denotacional completa de PCF (sin error)

$$
\begin{aligned}
[\![x]\!]_\theta &= \theta(x)\\
[\![\lambda x : A.t]\!]_\theta &= a^{\in [\![A]\!]} \mapsto [\![t]\!]_{\theta, x=a}\\
[\![tr]\!]_\theta &= [\![t]\!]_\theta [\![r]\!]_\theta\\
[\![n]\!]_\theta &= n\\
[\![t \otimes r]\!]_\theta &= [\![t]\!]_\theta \otimes [\![r]\!]_\theta\\
[\![\mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s]\!]_\theta &= \begin{cases} [\![r]\!]_\theta & \text{si } [\![t]\!]_\theta = 0\\ [\![s]\!]_\theta & \text{si } [\![t]\!]_\theta \in \mathbb{N}^*\\ \bot_A & \text{si } [\![t]\!]_\theta = \bot_{\mathbb{N}} \text{ y } A \text{ es el tipo del término} \end{cases}\\
[\![\mathsf{let}\ x : A = r\ \mathsf{in}\ t]\!]_\theta &= [\![t]\!]_{\theta, x=[\![r]\!]_\theta}\\
[\![\mu x : A.t]\!]_\theta &= \mathsf{FIX}(a^{\in [\![A]\!]} \mapsto [\![t]\!]_{\theta, x=a})
\end{aligned}
$$

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

Observación. El mínimo de $[\![A]\!]$ es $\bot_A$ y el mínimo de $[\![A \Rightarrow B]\!] = \bot_{[\![A]\!] \to [\![B]\!]}$ que es la función constante $\bot_{[\![B]\!]}$.

Teorema 9.5. Si $\Gamma \vdash t : A$, entonces para toda valuación θ válida en Γ se tiene $[\![t]\!]_\theta \in [\![A]\!]$.

Demostración. Procedemos por inducción en la relación de tipado.

- $\Gamma, x : A \vdash x : A$. $[\![x]\!]_\theta = \theta(x) \in [\![A]\!]$.
- $\Gamma \vdash n : \mathsf{nat}$. $[\![n]\!]_\theta = n \in \mathbb{N}^\perp = [\![\mathsf{nat}]\!]$.
- $\dfrac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \Rightarrow B}$.

  $[\![\lambda x : A.t]\!]_\theta = a^{\in [\![A]\!]} \mapsto [\![t]\!]_{\theta, x=a}$. Por hipótesis de inducción $\forall a \in [\![A]\!], [\![t]\!]_{\theta, x=a} \in [\![B]\!]$, es decir, $a^{\in [\![A]\!]} \mapsto [\![t]\!]_{\theta, x=a} \in [\![A]\!] \to [\![B]\!] = [\![A \Rightarrow B]\!]$.
- $\dfrac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash r : A}{\Gamma \vdash tr : B}$

  $[\![tr]\!]_\theta = [\![t]\!]_\theta [\![r]\!]_\theta$. Por hipótesis de inducción $[\![t]\!]_\theta \in [\![A]\!] \to [\![B]\!]$ y $[\![r]\!]_\theta \in [\![B]\!]$, por lo tanto $[\![t]\!]_\theta [\![s]\!]_\theta \in [\![B]\!]$.
- $\dfrac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash r : \mathsf{nat}}{\Gamma \vdash t \otimes r : \mathsf{nat}}$

  $$[\![t \otimes r]\!]_\theta = \begin{cases} [\![t]\!]_\theta \otimes [\![r]\!]_\theta & \text{si } [\![t]\!]_\theta, [\![r]\!]_\theta \in \mathbb{N}\\ \bot_{\mathbb{N}} & \text{en otro caso} \end{cases}$$

  Por hipótesis de inducción $[\![t]\!]_\theta, [\![r]\!]_\theta \in \mathbb{N}^\perp$, entonces $[\![t]\!]_\theta \otimes [\![r]\!]_\theta \in \mathbb{N}^\perp = [\![\mathsf{nat}]\!]$.
- $\dfrac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash r : A \quad \Gamma \vdash s : A}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s : A}$

  $$[\![\mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s]\!]_\theta = \begin{cases} [\![r]\!]_\theta & \text{si } [\![t]\!]_\theta = 0\\ [\![s]\!]_\theta & \text{si } [\![t]\!]_\theta \in \mathbb{N}^*\\ \bot_A & \text{si } [\![t]\!]_\theta = \bot_{\mathbb{N}} \end{cases}$$

  Por hipótesis de inducción $[\![r]\!]_\theta \in [\![A]\!]$ y $[\![s]\!]_\theta \in [\![A]\!]$, entonces $[\![\mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s]\!]_\theta \in [\![A]\!]$.

- $\dfrac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{let}\ x : A = u\ \mathsf{in}\ t : B}$

  $[\![\mathsf{let}\ x = u\ \mathsf{in}\ t]\!]_\theta = [\![t]\!]_{\theta, x=[\![u]\!]_\theta}$

  Por hipótesis de inducción, para todo θ válido en Γ, $[\![u]\!]_\theta \in [\![A]\!]$, y para todo θ' válido en $\Gamma, x : A$, $[\![t]\!]_{\theta'} \in [\![B]\!]$. Dado que $[\![u]\!]_\theta \in [\![A]\!]$, tenemos que $\theta' = \theta, x = [\![u]\!]_\theta$ es una valuación válida en $\Gamma, x : A$, y por lo tanto $[\![t]\!]_{\theta, x=[\![u]\!]_\theta} \in [\![B]\!]$.

- $\dfrac{\Gamma, x : A \vdash t : A}{\Gamma \vdash \mu x : A.t : A}$

  $[\![\mu x : A.t]\!]_\theta = FIX(a^{\in [\![A]\!]} \mapsto [\![t]\!]_{\theta, x=a})$. Por hipótesis de inducción, si θ es válida en Γ, y $a \in [\![A]\!]$, $[\![t]\!]_{\theta, x=a} \in [\![A]\!]$. El punto fijo de una función de $[\![A]\!]$ en $[\![A]\!]$ es un elemento de $[\![A]\!]$. Por lo tanto, $[\![\mu x : A.t]\!]_\theta \in [\![A]\!]$. □

Teorema 9.6 (Soundness)**.** Si $\Gamma \vdash t : A$ y $t \to r$, entonces para toda valuación θ válida en Γ se tiene $[\![t]\!]_\theta = [\![r]\!]_\theta$.

Demostración. Ejercicio (Ayuda: Inducción sobre la relación $\to$). □

10. Dos demostraciones importantes

En esta secciones vamos a demostrar dos teoremas importantes para tipos simples.

10.1. Subject reduction

En esta sección vamos a demostrar el Teorema 4.4, visto en la Sección 4.3:

Teorema 4.4 (Subject reduction o conservación de tipos)
Si $\Gamma \vdash t : A$ y $t \to u$ entonces $\Gamma \vdash u : A$.

Para demostrar este teorema, primero necesitaremos un resultado auxiliar, que nos dice que si un término t tiene un tipo y se substituye una variable libre por un término del mismo tipo que dicha variable, el tipo de t no varía.

Lema 10.1 (Substitución)**.** Si $\Gamma, x : A \vdash u : B$ y $\Gamma \vdash t : A$, entonces $\Gamma \vdash u[t/x] : B$.

Demostración. Procedemos por inducción estructural sobre u.

$u = x$**.** Es decir que $\Gamma, x : A \vdash x : B$, entonces $B = A$, y dado que $x[t/x] = t$, el caso queda demostrado.

$u = y$**.** Es decir que $\Gamma, x : A \vdash y : B$. Pero como $x \neq y$, se puede demostrar por inducción en la relación de tipado que $\Gamma \vdash y : B$. Dado que $y[t/x] = y$, el caso queda demostrado.

$u = \lambda y : C.r$**.** Es decir que $\Gamma, x : A \vdash \lambda y : C.r : B$, por lo tanto $B = C \Rightarrow D$ y $\Gamma, y : C, x : A \vdash r : D$. Entonces, por hipótesis de inducción, $\Gamma, y : C \vdash r[t/x] : D$, y por la regla $\Rightarrow_i$, $\Gamma \vdash \lambda y : C.r[t/x] : B$. Dado que $\lambda y : C.r[t/x] = (\lambda y : C.r)[t/x]$, el caso queda demostrado.

$u = rs$. Es decir que $\Gamma, x : A \vdash rs : B$. Entonces $\Gamma, x : A \vdash r : C \Rightarrow B$ y $\Gamma, x : A \vdash s : C$, y por hipótesis de inducción, $\Gamma \vdash r[t/x] : C \Rightarrow B$ y $\Gamma \vdash s[t/x] : C$. Por la regla $\Rightarrow_e$, tenemos $\Gamma \vdash r[t/x]s[t/x] : B$. Notar que $r[t/x]s[t/x] = (rs)[t/x]$, por lo que el caso queda demostrado.

$u = n$. Es decir que $\Gamma, x : A \vdash n : B$, por lo tanto $B = \mathsf{nat}$ y $\Gamma \vdash n : \mathsf{nat}$. Notar que $n[t/x] = n$, por lo que el caso queda demostrado.

$u = r \otimes s$. Es decir que $\Gamma, x : A \vdash r \otimes s : B$, y entonces $B = \mathsf{nat}$ y tenemos $\Gamma, x : A \vdash r : \mathsf{nat}$ y $\Gamma, x : A \vdash s : \mathsf{nat}$. Entonces, por hipótesis de inducción $\Gamma \vdash r[t/x] : \mathsf{nat}$ y $\Gamma \vdash s[t/x] : \mathsf{nat}$. Por lo tanto, por regla $\otimes$, $\Gamma \vdash r[t/x] \otimes s[t/x] : \mathsf{nat}$. Dado que $r[t/x] \otimes s[t/x] = (r \otimes s)[t/x]$, el caso queda demostrado.

$u = \mathsf{ifz}\ r\ \mathsf{then}\ s\ \mathsf{else}\ o$. Es decir que $\Gamma, x : A \vdash \mathsf{ifz}\ r\ \mathsf{then}\ s\ \mathsf{else}\ o : B$, y entonces $\Gamma, x : A \vdash r : \mathsf{nat}$, $\Gamma, x : A \vdash s : B$ y $\Gamma, x : A \vdash o : B$. Por hipótesis de inducción, $\Gamma \vdash r[t/x] : \mathsf{nat}$, $\Gamma \vdash s[t/x] : B$ y $\Gamma \vdash o[t/x] : B$. Entonces, por regla ifz, $\Gamma \vdash \mathsf{ifz}\ r[t/x]\ \mathsf{then}\ s[t/x]\ \mathsf{else}\ o[t/x] : B$. Dado que $\mathsf{ifz}\ r[t/x]\ \mathsf{then}\ s[t/x]\ \mathsf{else}\ o[t/x] = (\mathsf{ifz}\ r\ \mathsf{then}\ s\ \mathsf{else}\ o)[t/x]$, el caso queda demostrado.

$u = \mu y : C.r$. Es decir que $\Gamma, x : A \vdash \mu y : C.r : B$, y entonces $C = B$ y $\Gamma, x : A, y : B \vdash r : B$. Por hipótesis de inducción, $\Gamma, y : B \vdash r[t/x] : B$. Entonces, por regla fix, $\Gamma \vdash \mu y : B.r[t/x] : B$. Dado que $\mu y : B.r[t/x] = (\mu y : C.r)[t/x]$ el caso queda demostrado.

$u = \mathsf{let}\ y : C = r\ \mathsf{in}\ s$. Es decir que $\Gamma, x : A \vdash \mathsf{let}\ y : C = r\ \mathsf{in}\ s : B$, y entonces $\Gamma, y : C, x : A \vdash s : B$ y $\Gamma, x : A \vdash r : C$. Por hipótesis de inducción $\Gamma, y : C \vdash s[t/x] : B$ y $\Gamma \vdash r[t/x] : C$, y entonces por la regla let, $\Gamma \vdash \mathsf{let}\ y : C = r[t/x]\ \mathsf{in}\ s[t/x] : B$. Dado que $\mathsf{let}\ y : C = r[t/x]\ \mathsf{in}\ s[t/x] = (\mathsf{let}\ y : C = r\ \mathsf{in}\ s)[t/x]$, el caso queda demostrado. □

Con este lema ya estamos en condiciones de probar el Teorema 4.4.

Demostración del Teorema 4.4. Procedemos por inducción sobre la relación $\to$.

- Casos base:

  $(\lambda x : B.u)t \to u[t/x]$. Entonces $\Gamma \vdash (\lambda x : B.u)t : A$, por lo tanto $\Gamma \vdash \lambda x : B.u : B \Rightarrow A$ y $\Gamma \vdash t : B$, y entonces $\Gamma, x : B \vdash u : A$. Entonces, por el Lema 10.1, $\Gamma \vdash u[t/x] : A$.

  $p \otimes q \to n$ **si** $p \otimes q = n$. Entonces $\Gamma \vdash p \otimes q : A$, por lo tanto $A = \mathsf{nat}$, y por la regla ax_c, $\Gamma \vdash n : \mathsf{nat}$.

  $\mathsf{ifz}\ 0\ \mathsf{then}\ t\ \mathsf{else}\ u \to t$. Entonces $\Gamma \vdash \mathsf{ifz}\ 0\ \mathsf{then}\ t\ \mathsf{else}\ u : A$, por lo que $\Gamma \vdash t : A$.

  $\mathsf{ifz}\ 1\ \mathsf{then}\ t\ \mathsf{else}\ u \to u$. Entonces $\Gamma \vdash \mathsf{ifz}\ 1\ \mathsf{then}\ t\ \mathsf{else}\ u : A$, por lo que $\Gamma \vdash u : A$.

  $\mu x.t \to t[\mu x.t/x]$. Entonces $\Gamma \vdash \mu x.t : A$, por lo tanto $\Gamma, x : A \vdash t : A$. Entonces, por Lema 10.1, $\Gamma \vdash t[\mu x.t/x] : A$.

  $\mathsf{let}\ x : B = t\ \mathsf{in}\ u \to u[t/x]$. Entonces $\Gamma \vdash \mathsf{let}\ x : B = t\ \mathsf{in}\ u : A$, por lo que $\Gamma, x : B \vdash u : A$ y $\Gamma \vdash t : B$. Entonces, por Lema 10.1, $\Gamma \vdash u[t/x] : A$.

- Casos inductivos. Supongamos que $t \to u$. Entonces:

  $\lambda x : B.t \to \lambda x : B.u$**.** Entonces $\Gamma \vdash \lambda x : B.t : A$, por lo tanto $A = B \Rightarrow C$ y $\Gamma, x : B \vdash t : C$. Por hipótesis de inducción $\Gamma, x : B \vdash u : C$, entonces por regla $\Rightarrow_i$, $\Gamma \vdash \lambda x : B.u : B \Rightarrow C = A$.

  $tr \to ur$**.** Entonces $\Gamma \vdash tr : A$, por lo tanto $\Gamma \vdash t : B \Rightarrow A$ y $\Gamma \vdash r : B$. Por hipótesis de inducción $\Gamma \vdash u : B \Rightarrow A$, entonces por regla $\Rightarrow_e$, $\Gamma \vdash ur : A$.

  $rt \to ru$**.** Entonces $\Gamma \vdash rt : A$, por lo tanto $\Gamma \vdash r : B \Rightarrow A$ y $\Gamma \vdash t : B$. Por hipótesis de inducción $\Gamma \vdash u : B$, entonces por regla $\Rightarrow_e$, $\Gamma \vdash ru : A$.

  $t \otimes r \to u \otimes r$**.** Entonces $\Gamma \vdash t \otimes r : A$, por lo tanto $A = \mathsf{nat}$, $\Gamma \vdash t : \mathsf{nat}$ y $\Gamma \vdash r : \mathsf{nat}$. Por hipótesis de inducción $\Gamma \vdash u : \mathsf{nat}$, entonces por regla $\otimes$, $\Gamma \vdash u \otimes r : \mathsf{nat} = A$.

  $r \otimes t \to r \otimes u$**.** Análogo al caso anterior.

  $\mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s \to \mathsf{ifz}\ u\ \mathsf{then}\ r\ \mathsf{else}\ s$**.** Entonces $\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s : A$, por lo tanto $\Gamma \vdash t : \mathsf{nat}$, $\Gamma \vdash r : A$ y $\Gamma \vdash s : A$. Por hipótesis de inducción $\Gamma \vdash u : \mathsf{nat}$, entonces por regla ifz, $\Gamma \vdash \mathsf{ifz}\ u\ \mathsf{then}\ r\ \mathsf{else}\ s : A$.

  $\mu x : B.t \to \mu x : B.u$**.** Entonces $\Gamma \vdash \mu x : B.t : A$, por lo tanto $B = A$ y $\Gamma, x : A \vdash t : A$. Por hipótesis de inducción $\Gamma, x : A \vdash u : A$, entonces por regla fix, $\Gamma \vdash \mu x : A.t : A$.

  $\mathsf{let}\ x : B = t\ \mathsf{in}\ r \to \mathsf{let}\ x : B = u\ \mathsf{in}\ r$**.** Entonces $\Gamma \vdash \mathsf{let}\ x : B = t\ \mathsf{in}\ r : A$, por lo tanto $\Gamma, x : B \vdash r : A$ y $\Gamma \vdash t : B$. Por hipótesis de inducción $\Gamma \vdash u : B$, entonces por regla let, $\Gamma \vdash \mathsf{let}\ x : B = u\ \mathsf{in}\ r : A$.

  $\mathsf{let}\ x : B = r\ \mathsf{in}\ t \to \mathsf{let}\ x : B = r\ \mathsf{in}\ u$**.** Entonces $\Gamma \vdash \mathsf{let}\ x : B = r\ \mathsf{in}\ t : A$, por lo tanto $\Gamma, x : B \vdash r : A$ y $\Gamma \vdash t : B$. Por hipótesis de inducción $\Gamma, x : B \vdash u : A$, entonces por regla let, $\Gamma \vdash \mathsf{let}\ x : B = r\ \mathsf{in}\ u : A$. □

10.2. Normalización fuerte

En esta sección vamos a demostrar el Teorema 4.5:

Teorema 4.2 (Teorema de Tait o normalización fuerte)
Todo término tipado que no contenga a fix, termina.

Esta demostración utiliza el concepto de semántica denotacional. Dado que vamos a probar que todos los programas terminan, no necesitamos agregar $\bot$ a los conjuntos. Simplemente interpretamos los tipos como conjuntos de términos fuertemente normalizantes, y luego interpretaremos los términos como elementos de esos conjuntos.
Sea FN el conjunto de todos los términos fuertemente normalizantes.

Definición 10.2. La interpretación de los tipos es la siguiente:

$$[\![\mathsf{nat}]\!] = \mathsf{FN}$$
$$[\![A \Rightarrow B]\!] = \{t \mid \forall r \in [\![A]\!], tr \in [\![B]\!]\}$$

Es decir, al tipo nat lo interpretamos en el conjunto de todos los términos que normalizan fuertemente y al tipo $A \Rightarrow B$, en el conjunto de términos que aceptan como argumento un término de $[\![A]\!]$ y devuelven un término de $[\![B]\!]$.

Lema 10.3. $\forall A, [\![A]\!] \subseteq \mathsf{FN}$.

Demostración. Procedemos por inducción sobre A.

- Si $A = \mathsf{nat}$, entonces $[\![\mathsf{nat}]\!] = \mathsf{FN} \subseteq \mathsf{FN}$.
- Si $A = B \Rightarrow C$, entonces para todo $t \in [\![A]\!]$ se tiene que para todo $r \in [\![B]\!]$, $tr \in [\![C]\!]$. Por hipótesis de inducción sobre $[\![C]\!]$, $tr \in \mathsf{FN}$ y por lo tanto $t \in \mathsf{FN}$. □

Definición 10.4 (Términos neutrales)**.** A los términos de la siguiente gramática se les llama términos neutrales:

$$N = tt \mid \mathsf{let}\ x : A = t\ \mathsf{in}\ t \mid \mathsf{ifz}\ t\ \mathsf{then}\ t\ \mathsf{else}\ t$$

Para cualquier término t, notamos por $|t|$ a la suma de nodos en el árbol formado por todos los caminos de reducción que comienzan en t.

Lema 10.5. Si todas las reducciones de un término neutral N están en $[\![A]\!]$, entonces $N \in [\![A]\!]$.

Demostración. Procedemos por inducción en A.

- Si $A = \mathsf{nat}$ entonces tenemos que mostrar que si todas las reducciones de N están en $[\![\mathsf{nat}]\!] = \mathsf{FN}$, entonces N está en FN, lo cual es cierto por definición.
- Si $A = B \Rightarrow C$ y todas las reducciones de N están en $[\![B \Rightarrow C]\!]$, entonces si $N \to t$, tenemos que para todo $r \in [\![B]\!]$, $tr \in [\![C]\!]$. Queremos mostrar que $Nr \in [\![C]\!]$. Por Lema 10.3, $r \in \mathsf{FN}$. Por lo tanto procedemos por inducción sobre $|r|$ para mostrar que todas las reducciones de Nr están en $[\![C]\!]$, y por lo tanto, por hipótesis de inducción $Nr \in [\![C]\!]$.

  Las reducciones posibles de Nr son las siguientes

  - tr con $N \to t$, y ya tenemos por suposición que $tr \in [\![C]\!]$.
  - Nr' con $r \to r'$, entonces dado que $|r'| < |r|$, podemos usar la hipótesis de inducción y concluir que $Nr' \in [\![C]\!]$.

  Dado que N es un término neutral, N no es de la forma $\lambda x : D.s$ y por lo tanto esas son las únicas reducciones posibles.

  Entonces, como $Nr \in [\![C]\!]$, tenemos que $N \in [\![B \Rightarrow C]\!]$. □

Lema 10.6. $\forall A, \forall x,\ x \in [\![A]\!]$.

Demostración. Procedemos por inducción sobre A.

- Si $A = \mathsf{nat}$, entonces claramente $x \in [\![\mathsf{nat}]\!] = \mathsf{FN}$.
- Si $A = B \Rightarrow C$, entonces tenemos que mostrar que $\forall r \in [\![B]\!]$, $xr \in [\![C]\!]$. Dado que xr es neutral, gracias al Lema 10.5 nos basta con demostrar que todas las reducciones en un paso de xr están en $[\![C]\!]$.

  Por Lema 10.3, $|r|$ existe (no es infinito), y por lo tanto, podemos procedemos por inducción sobre $|r|$.

Si $|r| = 0$, entonces xr no reduce y por lo tanto el conjunto de reducciones de xr es el conjunto vacío, que está trivialmente en $[\![C]\!]$.

Si $|r| > 0$, entonces $xr \to xr'$ son las únicas reducciones posibles, y como $|r| > |r'|$, podemos utilizar la hipótesis de inducción que nos dice que $xr' \in [\![C]\!]$.

Por lo tanto, todas las reducciones de xr están en $[\![C]\!]$, y el Lema 10.5, nos asegura que $xr \in [\![C]\!]$. Es decir, $x \in [\![B \Rightarrow C]\!]$.

□

El Lema 10.8 es análogo al Teorema 9.5, y su demostración también es análoga, sin embargo la reproducimos aquí por completitud. Para ello debemos redefinir el concepto de valuación y valuación válida.

Definición 10.7. Una valuación θ es una función que a cada variable de tipo A le asigna un elemento del conjunto $[\![A]\!]$. Además, usamos la notación $\theta(t)$ para un término t cualquiera para representar $t[\theta(x_1)/x_1][\theta(x_2)/x_2]\dots[\theta(x_n)/x_n]$, donde $FV(t) = \{x_1, x_2, \dots, x_n\}$.
Decimos que una valuación θ es válida en un contexto Γ, si para todo $x : A \in \Gamma$ se tiene $\theta(x) \in [\![A]\!]$.

Lema 10.8 (Adecuación)**.** Si $\Gamma \vdash t : A$ y θ es una valuación válida en Γ, entonces $\theta(t) \in [\![A]\!]$.

Demostración. Procedemos por inducción en la relación de tipado.

- $\Gamma, x : A \vdash x : A$.

  $\theta(x) \in [\![A]\!]$ por definición.
- $\Gamma \vdash n : \mathsf{nat}$.

  $\theta(n) = n \in \mathsf{FN} = [\![\mathsf{nat}]\!]$.
- $$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x : A.t : A \Rightarrow B}$$
  $\theta(\lambda x : A.t) = \lambda x : A.\theta(t)$.

  Quiero mostrar que $\theta(\lambda x : A.t) \in [\![A \Rightarrow B]\!]$, es decir que $\forall r \in [\![A]\!]$, tenemos $(\lambda x : A.\theta(t))r \in [\![B]\!]$. Procedemos por inducción en $|\theta(t)| + |r|$, donde $|s|$ es la cantidad de pasos del camino más largo para llegar a la forma normal de s. Como $\theta(t)$ y r son FN, podemos hacer esa inducción.

  Las reducciones posibles de $(\lambda x : A.\theta(t))r$ son:

  - $\theta(t)[r/x]$. Por hipótesis de inducción, si tomamos $\theta' = \theta, x = r$ tenemos $\theta'(t) \in [\![B]\!]$.
  - $(\lambda x : A.t')r$ con $\theta(t) \to t'$. Como $|t'| + |r| < |\theta(t)| + |r|$, la hipótesis de inducción aplica y podemos concluir que $(\lambda x : A.t')r \in [\![B]\!]$.
  - $(\lambda x : A.\theta(t))r'$ con $r \to r'$. Como $|\theta(t)| + |r'| < |\theta(t)| + |r|$, la hipótesis de inducción aplica y podemos concluir que $(\lambda x : A.\theta(t))r' \in [\![B]\!]$.

  Por lo tanto, todas las reducciones de $(\lambda x : A.\theta(t))r$ están en $[\![B]\!]$, y entonces, por Lema 10.5, $(\lambda x : A.\theta(t))r \in [\![B]\!]$.

- $\dfrac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash r : A}{\Gamma \vdash tr : B}$

  Por hipótesis de inducción, $\theta(t) \in [\![A \Rightarrow B]\!]$ y $\theta(r) \in [\![A]\!]$. Por lo tanto, por definición de $[\![A \Rightarrow B]\!]$ tenemos $\theta(t)\theta(r) \in [\![B]\!]$. Nótese que $\theta(t)\theta(r) = \theta(tr)$.
- $\dfrac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash r : \mathsf{nat}}{\Gamma \vdash t \otimes r : \mathsf{nat}}$

  Por hipótesis de inducción $\theta(t) \in \mathsf{FN}$ y $\theta(r) \in \mathsf{FN}$, por lo tanto $\theta(t) \otimes \theta(r) = \theta(t \otimes r) \in \mathsf{FN} = [\![\mathsf{nat}]\!]$.
- $\dfrac{\Gamma \vdash t : \mathsf{nat} \quad \Gamma \vdash r : A \quad \Gamma \vdash s : A}{\Gamma \vdash \mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s : A}$

  Por hipótesis de inducción $\theta(t) \in \mathsf{FN}$, $\theta(r) \in [\![A]\!]$ y $\theta(s) \in [\![A]\!]$. Procedemos por inducción sobre $|\theta(t)|$ para mostrar que todos los reductos de $\mathsf{ifz}\ \theta(t)\ \mathsf{then}\ \theta(r)\ \mathsf{else}\ \theta(s)$ están en $[\![A]\!]$.

  Los reductos son:

  - $\mathsf{ifz}\ t'\ \mathsf{then}\ \theta(r)\ \mathsf{else}\ \theta(s)$ con $\theta(t) \to t'$. Entonces $t' \in \mathsf{FN}$ y $|t'| < |\theta(t)|$. Por lo tanto, la hipótesis de inducción aplica y $\mathsf{ifz}\ t'\ \mathsf{then}\ \theta(r)\ \mathsf{else}\ \theta(s) \in [\![A]\!]$.
  - $\theta(r)$ si $\theta(t) = 0$, y por hipótesis $\theta(r) \in [\![A]\!]$.
  - $\theta(s)$ si $\theta(t) = n \neq 0$, y por hipótesis $\theta(s) \in [\![A]\!]$.

  Por lo tanto, por Lema 10.5, $\theta(\mathsf{ifz}\ t\ \mathsf{then}\ r\ \mathsf{else}\ s) = \mathsf{ifz}\ \theta(t)\ \mathsf{then}\ \theta(r)\ \mathsf{else}\ \theta(s) \in [\![A]\!]$.
- $\dfrac{\Gamma, x : A \vdash t : B \quad \Gamma \vdash u : A}{\Gamma \vdash \mathsf{let}\ x : A = u\ \mathsf{in}\ t : B}$

  Por hipótesis de inducción, para todo θ válido en Γ, $\theta(u) \in [\![A]\!]$, y para todo θ' válido en $\Gamma, x : A$, $\theta'(t) \in [\![B]\!]$. Dado que $\theta(u) \in [\![A]\!]$, tenemos que $\theta' = \theta, x = \theta(u)$ es una valuación válida en $\Gamma, x : A$, y por lo tanto $\theta'(t) \in [\![B]\!]$. Procedemos por inducción sobre $|\theta(t)| + |\theta(u)|$ para mostrar que todos los reductos de $\mathsf{let}\ x : A = \theta(u)\ \mathsf{in}\ \theta(t)$ están en $[\![B]\!]$.

  Los reductos son:

  - $\mathsf{let}\ x : A = u'\ \mathsf{in}\ \theta(t)$ con $\theta(u) \to u'$. Entonces la hipótesis de inducción aplica.
  - $\mathsf{let}\ x : A = \theta(u)\ \mathsf{in}\ t'$ con $\theta(t) \to t'$. Entonces la hipótesis de inducción aplica.
  - $\theta(t)[\theta(u)/x] = \theta'(t) \in [\![B]\!]$.

  Por lo tanto, por Lema 10.5, $\theta(\mathsf{let}\ x : A = u\ \mathsf{in}\ t) = \mathsf{let}\ x : A = \theta(u)\ \mathsf{in}\ \theta(t) \in [\![A]\!]$. □

Demostración del Teorema 4.5 (Teorema de Tait). Sea $\Gamma \vdash t : A$ donde t no contiene fix. Por el Lema 10.6, $\theta = \mathsf{id}$ es una valuación válida en Γ. Por lo tanto, por el Lema 10.8, tenemos $t \in [\![A]\!]$, y por el Lema 10.3, $t \in \mathsf{FN}$. □