

# Cálculo Lambda Tipado (2/3)

Alejandro Ríos

Departamento de Computación, FCEyN, UBA

*“There may, indeed, be other applications of the system  
other than its use as a logic”*

Alonzo Church, 1932

31 de agosto de 2017

# La clase pasada

- ▶ Cálculo Lambda tipado y extensiones
  - ▶ Funciones, aplicación
  - ▶ Expresiones booleanas
  - ▶ Expresiones aritméticas
  - ▶ Unit
  - ▶ Declaraciones locales
  - ▶ Registros
- ▶ Para cada extensión
  - ▶ Expresiones de tipos
  - ▶ Términos
  - ▶ Tipado
  - ▶ Valores
  - ▶ Semántica operacional small-step

# La clase pasada

$$\text{Corrección} = \text{Progreso} + \text{Preservación}$$

## Progreso

Si  $M$  es cerrado y bien tipado entonces

1.  $M$  es un valor
2. o bien existe  $M'$  tal que  $M \rightarrow M'$

*La evaluación no puede trabarse para términos cerrados, bien tipados que no son valores.*

## Preservación

Si  $\Gamma \triangleright M : \sigma$  y  $M \rightarrow N$ , entonces  $\Gamma \triangleright N : \sigma$

*La evaluación preserva tipos.*

## Terminación

La evaluación de todo término bien tipado termina.

*La evaluación de los términos bien tipados no se cuelga.*

# Hoy - Dos extensiones más

- ▶ Referencias

Programación Imperativa = Progr. Funcional + Efectos

- ▶ Recursión

- ▶ Ninguna de las extensiones vistas permite definir funciones recursivas.
- ▶ Todas las funciones definibles hasta el momento son totales.

# Referencias - Motivación

- ▶ En una expresión como  $let\ x : Nat = \underline{2}\ in\ M$ 
  - ▶  $x$  es una variable declarada con valor 2.
  - ▶ El valor de  $x$  permanece **inalterado** a lo largo de la evaluación de  $M$ .
  - ▶ En este sentido  $x$  es **immutable**: no existe una operación de asignación.
- ▶ En programación imperativa pasa todo lo **contrario**.
  - ▶ **Todas** las variables son **mutables**.
- ▶ Vamos a extender Cálculo Lambda Tipado con variables mutables.

# Operaciones básicas

## Alocación (Reserva de memoria)

$\text{ref } M$  genera una referencia fresca cuyo contenido es el valor de  $M$ .

## Derreferenciación (Lectura)

$!x$  sigue la referencia  $x$  y retorna su contenido.

## Asignación

$x := M$  almacena en la referencia  $x$  el valor de  $M$ .

# Ejemplos

Nota: En los ejemplos de esta clase omitiremos los tipos de las let-expresiones para facilitar la lectura.

- ▶  $\text{let } x = \text{ref } \underline{2} \text{ in } (\lambda\_ : \text{unit}.!x) (x := \text{succ}(!x))$  evalúa a  $\underline{3}$ .
- ▶ ¿ $\text{let } x = \text{ref } \underline{2} \text{ in } x$  a qué evalúa?
- ▶  $\text{let } x = \underline{2} \text{ in } x$  evalúa a  $\underline{2}$ .
- ▶  $\text{let } x = \text{ref } \underline{2} \text{ in let } y = x \text{ in } (\lambda\_ : \text{unit}.!x) (y := \text{succ}(!y))$  evalúa a  $\underline{3}$ .
  - ▶  $x$  e  $y$  son **alias** para la misma celda de memoria.

# Comandos = Expresiones con efectos

- ▶ El término *let  $x = \text{ref } \underline{2}$  in  $x := \text{succ}(!x)$* , ¿A qué evalúa?
- ▶ La asignación es una expresión que interesa por su **efecto** y **no** su valor.
  - ▶ No tiene interés preguntarse por el **valor** de una asignación.
  - ▶ ¡Sí tiene sentido preguntarse por el **efecto**!

## Comando

Expresión que se evalúa para causar un efecto; definimos a *unit* como su valor.

- ▶ Un lenguaje funcional **puro** es uno en el que las expresiones son **puras** en el sentido de carecer de efectos.



# Expresiones de tipos

Las expresiones de tipos se extienden del siguiente modo

$$\sigma ::= Bool \mid Nat \mid \sigma \rightarrow \tau \mid Unit \mid Ref \sigma$$

Descripción informal:

- ▶  $Ref \sigma$  es el tipo de las referencias a valores de tipo  $\sigma$ .
- ▶ Ej.  $Ref (Bool \rightarrow Nat)$  es el tipo de las referencias a funciones de  $Bool$  en  $Nat$ .

# Términos

$$\begin{array}{lcl} M & ::= & x \\ & | & \lambda x : \sigma. M \\ & | & M N \\ & | & \textit{unit} \\ & | & \textit{ref } M \\ & | & !M \\ & | & M := N \\ & | & \dots \end{array}$$

El sistema de tipado **excluirá** términos “mal formados”.

- ▶ !2
- ▶ 2 := 3

# Reglas de tipado

- ▶ Las reglas de tipado serán presentadas en **dos etapas**.
- ▶ Primera presentación:
  - ▶ es de carácter **preliminar**;
  - ▶ se basa en la sintaxis de términos introducidas al momento.
- ▶ Segunda presentación:
  - ▶ es la **definitiva**;
  - ▶ al estudiar la semántica operacional surgirá la necesidad de **ampliar la sintaxis** por cuestiones técnicas;
  - ▶ se basa en la sintaxis de términos **ampliada**.

## Reglas de tipado - Preliminares

$$\frac{\Gamma \triangleright M_1 : \sigma}{\Gamma \triangleright \text{ref } M_1 : \text{Ref } \sigma} \text{ (T-REF)}$$

$$\frac{\Gamma \triangleright M_1 : \text{Ref } \sigma}{\Gamma \triangleright !M_1 : \sigma} \text{ (T-DEREF)}$$

$$\frac{\Gamma \triangleright M_1 : \text{Ref } \sigma_1 \quad \Gamma \triangleright M_2 : \sigma_1}{\Gamma \triangleright M_1 := M_2 : \text{Unit}} \text{ (T-ASSIGN)}$$

## Ejemplos

- ▶  $let\ x = ref\ \underline{2}\ in\ (\lambda\_ : unit.!x)\ (x := succ(!x))$
- ▶  $let\ x = ref\ \underline{2}\ in\ x$
- ▶  $let\ x = \underline{2}\ in\ x$
- ▶  $let\ x = ref\ \underline{2}\ in\ let\ y = x\ in\ (\lambda\_ : unit.!x)\ (y := succ(!y))$

Nota: el ítem del primer punto puede escribirse también:

$$let\ x = ref\ \underline{2}\ in\ (x := succ(!x)); !x$$

Recordemos que:

$$M_1; M_2 \stackrel{\text{def}}{=} (\lambda x : Unit.M_2)\ M_1 \quad x \notin FV(M_2)$$

# Motivación

Al intentar formalizar la semántica operacional surgen las preguntas:

- ▶ ¿Cuáles son los valores de tipo  $Ref\ \sigma$ ?
- ▶ ¿Cómo modelizar la evaluación del término  $ref\ M$ ?

Las respuestas dependen de otra pregunta.

¿Qué es una referencia?

**Rta.** Es una abstracción de una porción de memoria que se encuentra en uso.

# Memoria o “store”

- ▶ Usamos direcciones (simbólicas) o “locations”  $l, l_i \in \mathcal{L}$  para representar referencias.

Memoria (o “store”): función parcial de direcciones a valores.

- ▶ Usamos letras  $\mu, \mu'$  para referirnos a stores.
- ▶ Notación:
  - ▶  $\mu[l \mapsto V]$  es el store resultante de pisar  $\mu(l)$  con  $V$ .
  - ▶  $\mu \oplus (l \mapsto V)$  es el store extendido resultante de ampliar  $\mu$  con una nueva asociación  $l \mapsto V$  (asumimos  $l \notin \text{Dom}(\mu)$ ).

Los juicios de evaluación toman la forma:

$$M \mid \mu \rightarrow M' \mid \mu'$$

# Valores

Intuición:

$$\frac{I \notin \text{Dom}(\mu)}{\text{ref } V \mid \mu \rightarrow I \mid \mu \oplus (I \mapsto V)} \text{ (E-REFV)}$$

Los valores posibles ahora incluyen las **direcciones**.

$$V ::= \text{unit} \mid \lambda x : \sigma. M \mid I$$

Dado que los valores son un **subconjunto** de los términos,

- ▶ debemos ampliar los términos con **direcciones**;
- ▶ éstas son producto de la formalización y **no** se pretende que sean utilizadas por el programador.



# Términos extendidos

$$\begin{array}{lcl} M & ::= & x \\ & | & \lambda x : \sigma. M \\ & | & M N \\ & | & unit \\ & | & ref\ M \\ & | & !M \\ & | & M := N \\ & | & / \\ & | & \dots \end{array}$$

# Juicios de tipado

$$\Gamma \triangleright l : ?$$

- ▶ **Depende** de los valores que se almacenen en la dirección  $l$ .
- ▶ Situación parecida a las **variables libres**.
- ▶ Precisamos un “**contexto de tipado**” para direcciones:
  - ▶  $\Sigma$  función parcial de direcciones en tipos.

## Nuevo juicio de tipado

$$\Gamma | \Sigma \triangleright M : \sigma$$

## Reglas de tipado - Definitivas

$$\frac{\Gamma|\Sigma \triangleright M_1 : \sigma}{\Gamma|\Sigma \triangleright \text{ref } M_1 : \text{Ref } \sigma} \text{ (T-REF)}$$

$$\frac{\Gamma|\Sigma \triangleright M_1 : \text{Ref } \sigma}{\Gamma|\Sigma \triangleright !M_1 : \sigma} \text{ (T-DEREF)}$$

$$\frac{\Gamma|\Sigma \triangleright M_1 : \text{Ref } \sigma_1 \quad \Gamma|\Sigma \triangleright M_2 : \sigma_1}{\Gamma|\Sigma \triangleright M_1 := M_2 : \text{Unit}} \text{ (T-ASSIGN)}$$

$$\frac{\Sigma(l) = \sigma}{\Gamma|\Sigma \triangleright l : \text{Ref } \sigma} \text{ (T-LOC)}$$

## Juicios de evaluación en un paso

- ▶ Retomamos la semántica operacional.
- ▶ Vamos a introducir axiomas y reglas que permiten darle significado al juicio de evaluación en un paso.

$$M \mid \mu \rightarrow M' \mid \mu'$$

- ▶ Recordar el conjunto de valores (expresiones resultantes de evaluar por completo a términos cerrados y bien tipados).

$$V ::= true \mid false \mid 0 \mid \underline{n} \mid unit \mid \lambda x : \sigma. M \mid /$$

## Juicios de evaluación en un paso (1/4)

$$\frac{M_1 \mid \mu \rightarrow M'_1 \mid \mu'}{M_1 M_2 \mid \mu \rightarrow M'_1 M_2 \mid \mu'} \text{ (E-APP1)}$$

$$\frac{M_2 \mid \mu \rightarrow M'_2 \mid \mu'}{(\lambda x : \sigma.M) M_2 \mid \mu \rightarrow (\lambda x : \sigma.M) M'_2 \mid \mu'} \text{ (E-APP2)}$$

$$\frac{}{(\lambda x : \sigma.M) \mathbf{V} \mid \mu \rightarrow M\{x \leftarrow \mathbf{V}\} \mid \mu} \text{ (E-APPABS)}$$

**Nota:** Estas reglas no modifican el store.

## Juicios de evaluación en un paso (2/4)

$$\frac{M_1 \mid \mu \rightarrow M'_1 \mid \mu'}{!M_1 \mid \mu \rightarrow !M'_1 \mid \mu'} \text{ (E-DEREF)}$$

$$\frac{\mu(l) = \textcolor{red}{V}}{!l \mid \mu \rightarrow \textcolor{red}{V} \mid \mu} \text{ (E-DEREFLOC)}$$

## Juicios de evaluación en un paso (3/4)

$$\frac{M_1 \mid \mu \rightarrow M'_1 \mid \mu'}{M_1 := M_2 \mid \mu \rightarrow M'_1 := M_2 \mid \mu'} \text{ (E-ASSIGN1)}$$

$$\frac{M_2 \mid \mu \rightarrow M'_2 \mid \mu'}{\textcolor{red}{V} := M_2 \mid \mu \rightarrow \textcolor{red}{V} := M'_2 \mid \mu'} \text{ (E-ASSIGN2)}$$

$$\frac{}{l := \textcolor{red}{V} \mid \mu \rightarrow \textit{unit} \mid \mu[l \mapsto \textcolor{red}{V}]} \text{ (E-ASSIGN)}$$

## Juicios de evaluación en un paso (4/4)

$$\frac{M_1 \mid \mu \rightarrow M'_1 \mid \mu'}{\text{ref } M_1 \mid \mu \rightarrow \text{ref } M'_1 \mid \mu'} \text{ (E-REF)}$$

$$\frac{l \notin \text{Dom}(\mu)}{\text{ref } \textcolor{red}{V} \mid \mu \rightarrow l \mid \mu \oplus (l \mapsto \textcolor{red}{V})} \text{ (E-REFV)}$$



## Ejemplo

$let\ x = \text{ref}\ \underline{2}\ in\ (\lambda\_ : Unit.!x)\ (x := succ(!x)) \mid \mu$

$\rightarrow let\ x = l_1\ in\ (\lambda\_ : Unit.!x)\ (x := succ(!x)) \mid \mu \oplus (l_1 \mapsto \underline{2})$

$\rightarrow (\lambda\_ : Unit.!l_1)\ (l_1 := succ(!l_1)) \mid \mu \oplus (l_1 \mapsto \underline{2})$

$\rightarrow (\lambda\_ : Unit.!l_1)\ (l_1 := succ(\underline{2})) \mid \mu \oplus (l_1 \mapsto \underline{2})$

$\rightarrow (\lambda\_ : Unit.!l_1)\ unit \mid (\mu \oplus (l_1 \mapsto \underline{2}))[l_1 \mapsto \underline{3}]$

$\rightarrow !l_1 \mid \mu \oplus (l_1 \mapsto \underline{3})$

$\rightarrow \underline{3} \mid \mu \oplus (l_1 \mapsto \underline{3})$

# Ejemplo

Sea

$$\begin{aligned} M &= \lambda r : \text{Ref}(\text{Unit} \rightarrow \text{Unit}). \\ &\quad \text{let } f = !r \\ &\quad \text{in } (r := \lambda x : \text{Unit}.f\ x); (!r)\ \text{unit} \end{aligned}$$

$M(\text{ref } (\lambda x : \text{Unit}.x)) \mid \mu$   
→  $M\ l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : \text{Unit}.x)$   
→  $\text{let } f = !l_1 \text{ in } (l_1 := \lambda x : \text{Unit}.f\ x); (!l_1)\ \text{unit} \mid \dots$   
→  $\text{let } f = \lambda x : \text{Unit}.x \text{ in } (l_1 := \lambda x : \text{Unit}.f\ x); (!l_1)\ \text{unit} \mid \dots$   
→  $(l_1 := \lambda x : \text{Unit}.(\lambda x : \text{Unit}.x)\ x); (!l_1)\ \text{unit} \mid \dots$   
→  $\text{unit}; (!l_1)\ \text{unit} \mid \mu \oplus (l_1 \mapsto \lambda x : \text{Unit}.(\lambda x : \text{Unit}.x)\ x)$   
→  $(!l_1)\ \text{unit} \mid \mu \oplus (l_1 \mapsto \lambda x : \text{Unit}.(\lambda x : \text{Unit}.x)\ x)$   
→  $(\lambda x : \text{Unit}.(\lambda x : \text{Unit}.x)\ x)\ \text{unit} \mid \dots$   
→  $(\lambda x : \text{Unit}.x)\ \text{unit} \mid \dots$   
→  $\text{unit} \mid \dots$

## Ejemplo

Sea

$$\begin{aligned} M = & \lambda r : \text{Ref}(\text{Unit} \rightarrow \text{Unit}). \\ & \text{let } f = !r \\ & \text{in } (r := \lambda x : \text{Unit}. f\ x); (!r) \text{ unit} \end{aligned}$$

Reemplazamos  $f$  por  $!r$  y nos queda

$$\begin{aligned} M' = & \lambda r : \text{Ref}(\text{Unit} \rightarrow \text{Unit}). \\ & (r := \lambda x : \text{Unit}. (!r)\ x); (!r) \text{ unit} \end{aligned}$$

Vamos a evaluar este nuevo  $M'$  aplicado al mismo término que en el slide anterior y ver qué pasa...

## Ejemplo

$$M' = \lambda r : \text{Ref } (\text{Unit} \rightarrow \text{Unit}). \\ (r := \lambda x : \text{Unit}.(!r) x); (!r) \text{ unit}$$

$$\begin{aligned} & M' (\text{ref } (\lambda x : \text{Unit}.x)) \mid \mu \\ \rightarrow & M' l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : \text{Unit}.x) \\ \rightarrow & (l_1 := \lambda x : \text{Unit}.(!l_1) x); (!l_1) \text{ unit} \mid \dots \\ \rightarrow & \text{unit}; (!l_1) \text{ unit} \mid \mu \oplus (l_1 \mapsto \lambda x : \text{Unit}.(!l_1) x) \\ \rightarrow & \boxed{(!l_1) \text{ unit}} \mid \dots \\ \rightarrow & (\lambda x : \text{Unit}.(!l_1) x) \text{ unit} \mid \dots \\ \rightarrow & \boxed{(!l_1) \text{ unit}} \mid \dots \\ \rightarrow & \dots \end{aligned}$$

Nota: no todo término cerrado y bien tipado termina en  $\lambda^{bnr}$   
( $\lambda$ -cálculo con booleanos, naturales y referencias).

# La clase pasada - Corrección de sistema de tipos

$$\text{Corrección} = \text{Progreso} + \text{Preservación}$$

## Progreso

Si  $M$  es cerrado y bien tipado entonces

1.  $M$  es un valor
2. o bien existe  $M'$  tal que  $M \rightarrow M'$

## Preservación

Si  $\Gamma \triangleright M : \sigma$  y  $M \rightarrow N$ , entonces  $\Gamma \triangleright N : \sigma$

Debemos **reformular** estos resultados en el marco de referencias.

# Preservación - Formulación ingenua

La formulación ingenua siguiente es **errónea**:

$$\Gamma | \Sigma \triangleright M : \sigma \quad \text{y} \quad M | \mu \rightarrow M' | \mu' \quad \text{implica} \quad \Gamma | \Sigma \triangleright M' : \sigma$$

- ▶ **El problema**: puede que la semántica no respete los tipos asumidos por el sistema de tipos para las direcciones (i.e.  $\Sigma$ ).
- ▶ Vamos a ver un ejemplo concreto.

# Preservación - Formulación ingenua

$\Gamma | \Sigma \triangleright M : \sigma$  y  $M | \mu \rightarrow M' | \mu'$  implica  $\Gamma | \Sigma \triangleright M' : \sigma$

Supongamos que

- ▶  $M = !I$
- ▶  $\Gamma = \emptyset$
- ▶  $\Sigma(I) = \text{Nat}$
- ▶  $\mu(I) = \text{true}$

Observar que

- ▶  $\Gamma | \Sigma \triangleright M : \text{Nat}$  y
- ▶  $M | \mu \rightarrow \text{true} | \mu$
- ▶ pero  $\Gamma | \Sigma \triangleright \text{true} : \text{Nat}$  no vale.

## Preservación - Formulación ingenua

$$\Gamma | \Sigma \triangleright M : \sigma \quad \text{y} \quad M | \mu \rightarrow M' | \mu' \quad \text{implica} \quad \Gamma | \Sigma \triangleright M' : \sigma$$

Supongamos que

- ▶  $M = !I$
- ▶  $\Gamma = \emptyset$
- ▶  $\Sigma(I) = \boxed{Nat}$
- ▶  $\mu(I) = \boxed{true}$

Observar que

- ▶  $\Gamma | \Sigma \triangleright M : Nat$  y
- ▶  $M | \mu \rightarrow true | \mu$
- ▶ pero  $\Gamma | \Sigma \triangleright true : Nat$  no vale.



# Preservación - Reformulada

- Precisamos una noción de compatibilidad entre el store y el contexto de tipado para stores.
  - Debemos tipar los stores.

- Introducimos un nuevo “juicio de tipado”:

$$\Gamma | \Sigma \triangleright \mu$$

- Este juicio se define del siguiente modo:

$$\Gamma | \Sigma \triangleright \mu \text{ sii}$$

1.  $Dom(\Sigma) = Dom(\mu)$  y
2.  $\Gamma | \Sigma \triangleright \mu(l) : \Sigma(l)$  para todo  $l \in Dom(\mu)$ .

# Preservación - Reformulada

Reformulamos preservación del siguiente modo.

$$\text{Si } \Gamma | \Sigma \triangleright M : \sigma \text{ y } M | \mu \rightarrow N | \mu' \text{ y } \Gamma | \Sigma \triangleright \mu,$$
$$\text{entonces } \Gamma | \boxed{\Sigma} \triangleright N : \sigma.$$

- ▶ Esto es **casi** correcto.
- ▶ No contempla la posibilidad de que el  $\Sigma$  encuadrado haya crecido en dominio respecto a  $\Sigma$ .
  - ▶ Por posibles reservas de memoria.

# Preservación - Definitiva

Si

- ▶  $\Gamma | \Sigma \triangleright M : \sigma$
- ▶  $M | \mu \rightarrow N | \mu'$
- ▶  $\Gamma | \Sigma \triangleright \mu$

implica que existe  $\Sigma' \supseteq \Sigma$  tal que

- ▶  $\Gamma | \Sigma' \triangleright N : \sigma$
- ▶  $\Gamma | \Sigma' \triangleright \mu'$

## Progreso - Reformulado

Si  $M$  es cerrado y bien tipado (i.e.  $\emptyset \mid \Sigma \triangleright M : \sigma$  para algún  $\Sigma, \sigma$ ) entonces:

1.  $M$  es un valor
2. o bien para cualquier store  $\mu$  tal que  $\emptyset \mid \Sigma \triangleright \mu$ , existe  $M'$  y  $\mu'$  tal que  $M \mid \mu \rightarrow M' \mid \mu'$ .

# Recursión

Ecuación recursiva

$$f = \dots f \dots f \dots$$

Dos explicaciones de la función denotada (cuando existe).

- ▶ Denotacional
  - ▶ Límite de una cadena de aproximaciones
- ▶ Operacional
  - ▶ El “desdoblador” y puntos fijos

Nota: a desarrollar en el pizarrón.

## Términos y tipado

$$M ::= \dots \mid \text{fix } M$$

- No se precisan nuevos tipos pero sí una regla de tipado.

$$\frac{\Gamma \triangleright M : \sigma_1 \rightarrow \sigma_1}{\Gamma \triangleright \text{fix } M : \sigma_1} \text{ (T-FIX)}$$

## Semántica operacional small-step

No hay valores nuevos pero sí reglas de evaluación en un paso nuevas.

$$\frac{M_1 \rightarrow M'_1}{\text{fix } M_1 \rightarrow \text{fix } M'_1} \text{ (E-FIX)}$$

$$\frac{}{\text{fix } (\lambda x : \sigma. M) \rightarrow M\{x \leftarrow \text{fix } (\lambda x : \sigma. M)\}} \text{ (E-FIXBETA)}$$

# Ejemplos

Sea  $M$  el término

$\lambda f : \text{Nat} \rightarrow \text{Nat}.$

$\lambda x : \text{Nat}.$

$\text{if } \text{iszero}(x) \text{ then } \underline{1} \text{ else } x * f(\text{pred}(x))$

en

$\text{let } \text{fact} = \text{fix } M \text{ in } \text{fact } \underline{3}$



## Ejemplos

Ahora podemos definir funciones parciales:

$$\text{fix}(\lambda x : \text{Nat}. \text{succ } x)$$

# Ejemplos

Sea  $M$  el término

$\lambda s : \text{Nat} \rightarrow \text{Nat} \rightarrow \text{Nat}.$

$\lambda x : \text{Nat}.$

$\lambda y : \text{Nat}.$

*if* iszero( $x$ ) *then*  $y$  *else* succ( $s$  pred( $x$ )  $y$ )

en

*let* suma = fix  $M$  in suma23

## Letrec

Una construcción alternativa para definir funciones recursivas es

$$\textit{letrec } f : \sigma \rightarrow \sigma = \lambda x : \sigma. M \textit{ in } N$$

Por ejemplo,

$$\begin{aligned} &\textit{letrec} \\ &\quad \textit{fact} : \textit{Nat} \rightarrow \textit{Nat} = \\ &\quad \lambda x : \textit{Nat}. \textit{if } x = 0 \textit{ then } \underline{1} \textit{ else } x * \textit{fact}(\textit{pred}(x)) \\ &\quad \textit{in fact } \underline{3} \end{aligned}$$

*letrec* puede escribirse en términos de *fix* del siguiente modo:

$$\textit{let } f = \textit{fix}(\lambda f : \sigma \rightarrow \sigma. \lambda x : \sigma. M) \textit{ in } N$$

# Fin de la clase

La clase que viene...

...inferencia de tipos

...y algoritmo de unificación