

Cálculo Lambda Tipado (2)

Objetivo de un sistema de tipos

Garantizar la **ausencia** de estados de error

- ▶ Decimos que un término **termina** o que es **fuertemente normalizante** si no hay cadenas de reducción infinitas a partir de él.

Teorema

- ▶ Todo término bien tipado termina
- ▶ Si un término cerrado está bien tipado, entonces evalúa a un valor

Estado de error

- ▶ Estado (=término) que **no es** un valor pero en el que la evaluación está **trabada**
- ▶ Representa estado en el cual el sistema de run-time en una implementación real generaría una excepción

Ejemplos

- ▶ *if x then M else N*
 - ▶ Obs: no es cerrado
- ▶ *true M*
 - ▶ Obs: no es tipable



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

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

Tipos y términos de λ^{bn}

$$\sigma ::= Bool \mid Nat \mid \sigma \rightarrow \rho$$

$$M ::= \dots \mid 0 \mid succ(M) \mid pred(M) \mid iszero(M)$$

Descripción informal:

- ▶ $succ(M)$: evaluar M hasta arrojar un número e incrementarlo
- ▶ $pred(M)$: evaluar M hasta arrojar un número y decrementarlo
- ▶ $iszero(M)$: evaluar M hasta arrojar un número, luego retornar *true/false* según sea cero o no

Tipado de λ^{bn}

Agregamos a los axiomas y regla de tipado de λ^b los siguientes:

$$\frac{}{\Gamma \triangleright 0 : \text{Nat}} \text{ (T-ZERO)}$$

$$\frac{\Gamma \triangleright M : \text{Nat}}{\Gamma \triangleright \text{succ}(M) : \text{Nat}} \text{ (T-SUCC)}$$

$$\frac{\Gamma \triangleright M : \text{Nat}}{\Gamma \triangleright \text{pred}(M) : \text{Nat}} \text{ (T-PRED)}$$

$$\frac{\Gamma \triangleright M : \text{Nat}}{\Gamma \triangleright \text{iszero}(M) : \text{Bool}} \text{ (T-ISZERO)}$$

Valores y evaluación en un paso de λ^{bn} (1/2)

Valores

$V ::= \dots \mid \underline{n}$ donde \underline{n} abrevia $\text{succ}^n(0)$.

Juicio de evaluación en un paso (1/2)

$$\frac{M_1 \rightarrow M'_1}{\text{succ}(M_1) \rightarrow \text{succ}(M'_1)} \text{ (E-SUCC)}$$

$$\frac{}{\text{pred}(0) \rightarrow 0} \text{ (E-PREDZERO)}$$

$$\frac{}{\text{pred}(\underline{n+1}) \rightarrow \underline{n}} \text{ (E-PREDSUCC)}$$

$$\frac{M_1 \rightarrow M'_1}{\text{pred}(M_1) \rightarrow \text{pred}(M'_1)} \text{ (E-PRED)}$$

Valores y evaluación en un paso de $\lambda^{bn}(2/2)$

Juicio de evaluación en un paso (2/2)

$$\frac{}{iszero(0) \rightarrow true} \text{ (E-ISZEROZERO)}$$

$$\frac{}{iszero(\underline{n+1}) \rightarrow false} \text{ (E-ISZEROSUCC)}$$

$$\frac{M_1 \rightarrow M'_1}{iszero(M_1) \rightarrow iszero(M'_1)} \text{ (E-ISZERO)}$$

Además de los juicios de evaluación en un paso de $C\text{-}\lambda^b$.

Tipos y términos de $\lambda^{\dots r}$

Sea \mathcal{L} un conjunto de etiquetas

$$\sigma ::= \dots \mid \{l_i : \sigma_i \mid i \in 1..n\}$$

- ▶ $\{\text{nombre} : \text{String}, \text{edad} : \text{Nat}\}$
- ▶ $\{\text{persona} : \{\text{nombre} : \text{String}, \text{edad} : \text{Nat}\}, \text{cuil} : \text{Nat}\}$

$$\{\text{nombre} : \text{String}, \text{edad} : \text{Nat}\} \neq \{\text{edad} : \text{Nat}, \text{nombre} : \text{String}\}$$

Tipos y términos de $\lambda^{\dots r}$

$$M ::= \dots \mid \{l_i = M_i \mid i \in 1..n\} \mid M.l$$

Descripción informal:

- ▶ El registro $\{l_i = M_i \mid i \in 1..n\}$ evalúa a $\{l_i = V_i \mid i \in 1..n\}$ donde V_i es el valor al que evalúa M_i , $i \in 1..n$
- ▶ $M.l$: evaluar M hasta que arroje $\{l_i = V_i \mid i \in 1..n\}$, luego proyectar el campo correspondiente

Ejemplos

- ▶ $\lambda x : \text{Nat} . \lambda y : \text{Bool} . \{ \text{edad} = x, \text{esMujer} = y \}$
- ▶ $\lambda p : \{ \text{edad} : \text{Nat}, \text{esMujer} : \text{Bool} \} . p . \text{edad}$
- ▶ $(\lambda p : \{ \text{edad} : \text{Nat}, \text{esMujer} : \text{Bool} \} . p . \text{edad})$
 $\{ \text{edad} = 20, \text{esMujer} = \text{false} \}$

Tipado de $\lambda^{...r}$

$$\frac{\Gamma \triangleright M_i : \sigma_i \quad \text{para cada } i \in 1..n}{\Gamma \triangleright \{l_i = M_i \mid i \in 1..n\} : \{\sigma_i \mid i \in 1..n\}} \text{ (T-RCD)}$$

$$\frac{\Gamma \triangleright M : \{\sigma_i \mid i \in 1..n\} \quad j \in 1..n}{\Gamma \triangleright M.l_j : \sigma_j} \text{ (T-PROJ)}$$

Semántica operacional de λ^{cr}

Valores

$$V ::= \dots \mid \{l_i = V_i \mid i \in 1..n\}$$

Semántica operacional de $\lambda^{\dots r}$

$$\frac{M_j \rightarrow M'_j}{\{l_i = V_i \mid i \in 1..j-1, l_j = M_j, l_i = M_i \mid i \in j+1..n\} \rightarrow \{l_i = V_i \mid i \in 1..j-1, l_j = M'_j, l_i = M_i \mid i \in j+1..n\}} \text{ (E-RCD)}$$

$$\frac{j \in 1..n}{\{l_i = V_i \mid i \in 1..n\}.l_j \rightarrow V_j} \text{ (E-PROJCD)}$$

$$\frac{M \rightarrow M'}{M.l \rightarrow M'.l} \text{ (E-PROJ)}$$

Tipos y términos de λ^{bnu}

$$\sigma ::= Bool \mid Nat \mid Unit \mid \sigma \rightarrow \rho$$

$$M ::= \dots \mid unit$$

Descripción informal:

- ▶ *Unit* es un tipo unitario y el único valor posible de una expresión de ese tipo es *unit*.
- ▶ Cumple rol similar a *void* en C o Java

Tipado de λ^{bnu}

Agregamos el axioma de tipado:

$$\frac{}{\Gamma \triangleright unit : Unit} \text{ (T-UNIT)}$$

NB:

- ▶ No hay reglas de evaluación nuevas
- ▶ Extendemos el conjunto de valores V con $unit$

$$V ::= \dots \mid unit$$



- ▶ Su utilidad principal es en lenguajes con efectos laterales
- ▶ En estos lenguajes es útil poder evaluar varias expresiones en **secuencia**

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

- ▶ La evaluación de $M_1; M_2$ consiste en primero evaluar M_1 y luego M_2
- ▶ Con la definición dada, este comportamiento se logra con las reglas de evaluación definidas previamente

Tipos y términos de $\lambda^{\dots let}$

$$M ::= \dots \mid let\ x : \sigma = M\ in\ N$$

Descripción informal:

- ▶ $let\ x : \sigma = M\ in\ N$: evaluar M a un valor V , ligar x a V y evaluar N
- ▶ Mejora la legibilidad
- ▶ La extensión con let no implica agregar nuevos tipos

Ejemplo

- ▶ $\text{let } x : \text{Nat} = \underline{2} \text{ in succ}(x)$
- ▶ $\text{pred } (\text{let } x : \text{Nat} = \underline{2} \text{ in } x)$
- ▶ $\text{let } f : \text{Nat} \rightarrow \text{Nat} = \lambda x : \text{Nat}. \text{succ}(n) \text{ in } f(f0))$
- ▶ $\text{let } x : \text{Nat} = \underline{2} \text{ in let } x : \text{Nat} = \underline{3} \text{ in } x$

Tipado de $\lambda^{.../let}$

$$\frac{\Gamma \triangleright M : \sigma_1 \quad \Gamma, x : \sigma_1 \triangleright N : \sigma_2}{\Gamma \triangleright let\ x : \sigma_1 = M\ in\ N : \sigma_2} \text{(T-LET)}$$

Semántica operacional de $\lambda^{\dots/let}$

$$\frac{M_1 \rightarrow M'_1}{let\ x : \sigma = M_1\ in\ M_2 \rightarrow let\ x : \sigma = M'_1\ in\ M_2} \text{ (E-LET)}$$

$$\frac{}{let\ x : \sigma = V_1\ in\ M_2 \rightarrow M_2\{x \leftarrow V_1\}} \text{ (E-LETV)}$$

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.

- ▶ *let x = ref 2 in !x* evalúa a 2.
- ▶ *let x = ref 2 in ($\lambda_ : unit. !x$) ($x := succ(!x)$)* evalúa a 3.
- ▶ *let x = 2 in x* evalúa a 2.
- ▶ ¿*let x = ref 2 in x* a qué evalúa?
- ▶ *let x = ref 2 in let y = x in ($\lambda_ : unit. !x$) ($y := succ(!y)$)* evalúa a 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 σ .
- ▶ Ej. $Ref (Bool \rightarrow Nat)$ es el tipo de las referencias a funciones de $Bool$ en Nat .

Términos

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

Reglas de tipado - Preliminares

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

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

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

Motivación

Al intentar formalizar la semántica operacional surgen las preguntas:

- ▶ ¿Cuáles son los valores de tipo $Ref\ \sigma$?
- ▶ ¿Cómo modelar 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 μ, μ' 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 μ 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:

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

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

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

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:
 - ▶ Σ 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

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

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

Juicios de evaluación en un paso

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

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

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

Juicios de evaluación en un paso

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

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

Revisitar los juicios de evaluación para los restantes términos

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

$$\frac{M_2 \mid \mu \rightarrow M'_2 \mid \mu'}{\textcolor{red}{V}_1 M_2 \mid \mu \rightarrow \textcolor{red}{V}_1 M'_2 \mid \mu'} \text{ (E-APP2)}$$

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

Las de los restantes tipos (*Nat*, *Bool*,) son similares.

Nota: Estas reglas no modifican el store.

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. Σ).
- ▶ 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 Σ encuadrado haya crecido en dominio respecto a Σ .
 - ▶ 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 Σ, σ) entonces:

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

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 λ^{bnr}
(λ -cálculo con booleanos, naturales y referencias).

Recursión

Ecuación recursiva

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

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,

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

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

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