Cálculo Lambda Tipado (2)

Expresiones de tipos de λ^b

Las expresiones de tipos (o simplemente tipos) de λ^b son

$$\sigma, \tau \quad ::= \quad \mathit{Bool} \mid \sigma \to \tau$$

Descripción informal:

- *Bool* es el tipo de los booleanos,
- $\sigma \to \tau$ es el tipo de las funciones de σ en τ (donde σ y τ denotan expresiones de tipo)

Términos de λ^b

Sea $\mathcal{X}$ un conjunto infinito enumerable de variables y $x \in \mathcal{X}$. Los términos de λ^b están dados por

$$\begin{array}{rcl} M, N, P, Q & ::= & x \\ & \mid & \mathit{true} \\ & \mid & \mathit{false} \\ & \mid & \mathit{if}\ \ M\ \ \mathit{then}\ \ P\ \ \mathit{else}\ \ Q \\ & \mid & \lambda x : \sigma . M \\ & \mid & M\, N \end{array}$$

Resultados básicos

Unicidad de tipos

Si $\Gamma \rhd M : \sigma$ y $\Gamma \rhd M : \tau$ son derivables, entonces $\sigma = \tau$

Axiomas de tipado de λ^b

$$\frac{}{\Gamma \rhd \mathit{true} : \mathit{Bool}}\ (\textrm{T-}\textsc{True}) \qquad \frac{}{\Gamma \rhd \mathit{false} : \mathit{Bool}}\ (\textrm{T-}\textsc{False})$$

$$\frac{x : \sigma \in \Gamma}{\Gamma \rhd x : \sigma}\ (\textrm{T-}\textsc{Var})$$

Reglas de tipado de λ^b

$$\frac{\Gamma \rhd M : \mathit{Bool} \quad \Gamma \rhd P : \sigma \quad \Gamma \rhd Q : \sigma}{\Gamma \rhd \mathit{if}\ \ M\ \ \mathit{then}\, P\ \mathit{else}\ Q : \sigma}\ (\text{T-}\textsc{If})$$

$$\frac{\Gamma \rhd M : \sigma \to \tau \quad \Gamma \rhd N : \sigma}{\Gamma \rhd M\, N : \tau}\ (\text{T-}\textsc{App}) \qquad \frac{\Gamma, x : \sigma \rhd M : \tau}{\Gamma \rhd \lambda x : \sigma.M : \sigma \to \tau}\ (\text{T-}\textsc{Abs})$$

Semántica operacional de λ^b

Valores

$$V \quad ::= \quad \mathit{true} \mid \mathit{false} \mid \lambda x : \sigma.M$$

Introduciremos una noción de evalución en λ^b tal que valgan los lemas previos y también el siguiente resultado:

Teorema

Todo término bien-tipado y cerrado de tipo

- *Bool* evalúa, en cero o más pasos, a *true*, *false*
- $\sigma \to \tau$ evalúa, en cero o más pasos, a $\lambda x : \sigma.M$, para alguna variable x, para algún término M

Semántica Operacional - Expr. booleanas

Juicio de evaluación en un paso

$$\frac{}{\mathit{if}\ \mathit{true}\ \mathit{then}\ M_2\ \mathit{else}\ M_3 \to M_2}\ (\textsc{E-IfTrue})$$

$$\frac{}{\mathit{if}\ \mathit{false}\ \mathit{then}\ M_2\ \mathit{else}\ M_3 \to M_3}\ (\textsc{E-IfFalse})$$

$$\frac{M_1 \to M_1'}{\mathit{if}\ M_1\ \mathit{then}\ M_2\ \mathit{else}\ M_3 \to \mathit{if}\ M_1'\ \mathit{then}\ M_2\ \mathit{else}\ M_3}\ (\textsc{E-If})$$

Semántica operacional de λ^b

Juicio de evaluación en un paso

$$\frac{M_1 \to M_1'}{M_1\ M_2 \to M_1'\ M_2}\ (\textsc{E-App1}\ /\ \mu)$$

$$\frac{M_2 \to M_2'}{(\lambda x : \sigma.M)\ M_2 \to (\lambda x : \sigma.M)\ M_2'}\ (\textsc{E-App2}\ /\ \nu)$$

$$\frac{}{(\lambda x : \sigma.M)\ V \to M\{x \leftarrow V\}}\ (\textsc{E-AppAbs}\ /\ \beta)$$

Además de $(\textsc{E-IfTrue})$, $(\textsc{E-IfFalse})$, $(\textsc{E-If})$

Sustitución - Revisada

$$
\begin{array}{rcl}
x\{x \leftarrow N\} & \stackrel{\text{def}}{=} & N \\
a\{x \leftarrow N\} & \stackrel{\text{def}}{=} & a \quad \text{si } a \in \{\mathit{true}, \mathit{false}\} \cup \mathcal{X} \setminus \{x\} \\
(\mathit{if}\ M\ \mathit{then}\ P\ \mathit{else}\ Q)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & \mathit{if}\ M\{x \leftarrow N\}\ \mathit{then}\ P\{x \leftarrow N\} \\
 & & \qquad\qquad \mathit{else}\ Q\{x \leftarrow N\} \\
(M_1\, M_2)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & M_1\{x \leftarrow N\}\, M_2\{x \leftarrow N\} \\
(\lambda y : \sigma.M)\{x \leftarrow N\} & \stackrel{\text{def}}{=} & \lambda y : \sigma.M\{x \leftarrow N\}\ x \neq y,\ \ y \notin FV(N)
\end{array}
$$

1. NB: la condición $x \neq y,\ \ y \notin FV(N)$ siempre puede cumplirse renombrando apropiadamente
2. Técnicamente, la sustitución está definida sobre clases de α-equivalencia de términos

Propiedades

Una forma normal es un término que no puede evaluarse más (i.e. M tal que no existe N, $M \to N$)

Lema (Unicidad de formas normales)

Si $M \twoheadrightarrow U$ y $M \twoheadrightarrow V$ con U, V formas normales, entonces $U = V$

Lema (Terminación)

Para todo M existe una forma normal N tal que $M \twoheadrightarrow N$

Propiedades

Una forma normal es un término que no puede evaluarse más (i.e. M tal que no existe N, $M \to N$)

Lema (Unicidad de formas normales)

Si $M \twoheadrightarrow U$ y $M \twoheadrightarrow V$ con U, V formas normales, entonces $U = V$

Lema (Terminación)

Para todo M existe una forma normal N tal que $M \twoheadrightarrow N$

Lema

Todo valor está en forma normal

Propiedades

Una forma normal es un término que no puede evaluarse más (i.e. M tal que no existe N, $M \to N$)

Lema (Unicidad de formas normales)

Si $M \twoheadrightarrow U$ y $M \twoheadrightarrow V$ con U, V formas normales, entonces $U = V$

Lema (Terminación)

Para todo M existe una forma normal N tal que $M \twoheadrightarrow N$

Lema

Todo valor está en forma normal

- No vale el recíproco en λ^b (pero sí vale en el cálculo de las expresiones booleanas cerradas):
  - *if x then true else false*
  - x
  - *true false*

Objetivo de un sistema de tipos

Garantizar la ausencia de errores

Estado de error

- Un forma normal que no es un valor
- Representa un 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
  - Un término M es cerrado si $FV(M) = \emptyset$
- *true M*
  - Obs: no es tipable

Corrección

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

Progreso

Si M es cerrado y bien tipado entonces

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

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

Preservación

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

La evaluación preserva tipos

Tipos y términos de λ^{bn}

$$\sigma \quad ::= \quad Bool \mid Nat \mid \sigma \to \rho$$

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

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:

$$\frac{}{\Gamma \rhd 0 : Nat}\ (\text{T-Zero})$$

$$\frac{\Gamma \rhd M : Nat}{\Gamma \rhd succ(M) : Nat}\ (\text{T-Succ}) \qquad \frac{\Gamma \rhd M : Nat}{\Gamma \rhd pred(M) : Nat}\ (\text{T-Pred})$$

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

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

Valores

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

Juicio de evaluación en un paso (1/2)

$$\frac{M_1 \to M_1'}{succ(M_1) \to succ(M_1')} \text{(E-Succ)}$$

$$\frac{}{pred(0) \to 0} \text{(E-PredZero)} \qquad \frac{}{pred(\underline{n+1}) \to \underline{n}} \text{(E-PredSucc)}$$

$$\frac{M_1 \to M_1'}{pred(M_1) \to pred(M_1')} \text{(E-Pred)}$$

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

Juicio de evaluación en un paso (2/2)

$$\frac{}{\mathit{iszero}(0) \rightarrow \mathit{true}} \; (\text{E-IsZeroZero})$$

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

$$\frac{M_1 \rightarrow M_1'}{\mathit{iszero}(M_1) \rightarrow \mathit{iszero}(M_1')} \; (\text{E-IsZero})$$

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

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

Sea $\mathcal{L}$ un conjunto de etiquetas

$$\sigma \quad ::= \quad \ldots \mid \{l_i : \sigma_i \ ^{i \in 1..n}\}$$

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

Sea $\mathcal{L}$ un conjunto de etiquetas

$$\sigma \quad ::= \quad \ldots \mid \{l_i : \sigma_i \;^{i \in 1..n}\}$$

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

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

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

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

Descripción informal:

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

Ejemplos

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

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

$$\frac{\Gamma \rhd M_i : \sigma_i \quad \text{para cada } i \in 1..n}{\Gamma \rhd \{l_i = M_i \;^{i \in 1..n}\} : \{l_i : \sigma_i \;^{i \in 1..n}\}} \; (\text{T-Rcd})$$

$$\frac{\Gamma \rhd M : \{l_i : \sigma_i \;^{i \in 1..n}\} \quad j \in 1..n}{\Gamma \rhd M.l_j : \sigma_j} \; (\text{T-Proj})$$

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

Valores

$$V \quad ::= \quad \ldots \mid \{l_i = V_i \ ^{i \in 1..n}\}$$

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

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

$$\frac{M_j \to M_j'}{\begin{array}{c}\{l_i = V_i \; {}^{i\in 1..j-1}, l_j = M_j, l_i = M_i \; {}^{i\in j+1..n}\} \\ \to \\ \{l_i = V_i \; {}^{i\in 1..j-1}, l_j = M_j', l_i = M_i \; {}^{i\in j+1..n}\}\end{array}} \text{(E-RCD)}$$

$$\frac{j \in 1..n}{\{l_i = V_i \; {}^{i\in 1..n}\}.l_j \to V_j} \text{(E-PROJRCD)}$$

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

Tipos y términos de λ^{bnu}

$$\sigma \quad ::= \quad Bool \mid Nat \mid Unit \mid \sigma \to \rho$$

$$M \quad ::= \quad \ldots \mid unit$$

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:

$$\frac{}{\Gamma \rhd \mathit{unit} : \mathit{Unit}} \ (\textsc{T-Unit})$$

NB:

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

$$V \quad ::= \quad \ldots \mid \mathit{unit}$$

Utilidad

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

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

  - 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 $\lambda^{\ldots let}$

$$M \quad ::= \quad \ldots \mid let\ x : \sigma = M\ in\ N$$

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

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

Tipado de $\lambda^{\ldots let}$

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

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

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

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

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 inmutable: 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)

$ref\ M$ 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.

- $\mathit{let}\ x = \mathit{ref}\ \underline{2}\ \mathit{in}\ !x$ evalúa a

Ejemplos

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

- $\mathit{let}\ x = \mathit{ref}\ \underline{2}\ \mathit{in}\ !x$ evalúa a $\underline{2}$.
- $\mathit{let}\ x = \mathit{ref}\ \underline{2}\ \mathit{in}\ (\lambda_{\_} : \mathit{unit}.!x)\,(x := \mathit{succ}(!x))$ evalúa a

Ejemplos

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

- $\mathit{let}\ x = \mathit{ref}\ \underline{2}\ \mathit{in}\ !x$ evalúa a $\underline{2}$.
- $\mathit{let}\ x = \mathit{ref}\ \underline{2}\ \mathit{in}\ (\lambda\_ : \mathit{unit}.!x)\,(x := \mathit{succ}(!x))$ evalúa a $\underline{3}$.
- $\mathit{let}\ x = \underline{2}\ \mathit{in}\ x$ evalúa a

Ejemplos

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

- $let\ x = ref\ \underline{2}\ in\ !x$ evalúa a $\underline{2}$.
- $let\ x = ref\ \underline{2}\ in\ (\lambda_{\_} : unit.!x)\,(x := succ(!x))$ evalúa a $\underline{3}$.
- $let\ x = \underline{2}\ in\ x$ evalúa a $\underline{2}$.
- ¿$let\ x = ref\ \underline{2}\ in\ x$ a qué evalúa?

Ejemplos

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

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

Comandos = Expresiones con efectos

- El término $let\ x = ref\ \underline{2}\ in\ x := succ(!x)$, ¿A qué evalúa?

Comandos = Expresiones con efectos

- El término $let\ x = ref\ \underline{2}\ in\ x := succ(!x)$, ¿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

$$\sigma \quad ::= \quad \mathit{Bool} \mid \mathit{Nat} \mid \sigma \to \tau \mid \mathit{Unit} \mid \mathit{Ref}\ \sigma$$

Descripción informal:

- $\mathit{Ref}\ \sigma$ es el tipo de las referencias a valores de tipo σ.
- Ej. $\mathit{Ref}\ (\mathit{Bool} \to \mathit{Nat})$ es el tipo de las referencias a funciones de Bool en Nat.

Términos

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

Reglas de tipado - Preliminares

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

$$\frac{\Gamma \rhd M_1 : \mathit{Ref}\ \ \sigma}{\Gamma \rhd !M_1 : \sigma}\ (\text{T-DeRef})$$

$$\frac{\Gamma \rhd M_1 : \mathit{Ref}\ \ \sigma_1 \quad \Gamma \rhd M_2 : \sigma_1}{\Gamma \rhd M_1 := M_2 : \mathit{Unit}}\ (\text{T-Assign})$$

Motivación

Al intentar formalizar la semántica operacional surgen las preguntas:

- ¿Cuáles son los valores de tipo *Ref* σ?
- ¿Cómo modelar la evaluación del término *ref* M?

Las respuestas dependen de otra pregunta.

¿Qué es una referencia?

Motivación

Al intentar formalizar la semántica operacional surgen las preguntas:

- ¿Cuáles son los valores de tipo $\mathit{Ref}\ \sigma$?
- ¿Cómo modelar la evaluación del término $\mathit{ref}\ M$?

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 $l \notin Dom(\mu)$).

Los juicios de evaluación toman la forma:

$$M \mid \mu \to M' \mid \mu'$$

Valores

Intuición:

$$\frac{l \notin Dom(\mu)}{ref\ \ V \,|\, \mu \to l \,|\, \mu \oplus (l \mapsto V)}\ (\text{E-RefV})$$

Los valores posibles ahora incluyen las direcciones.

$$V \quad ::= \quad unit \,|\, \lambda x : \sigma.M \,|\, l$$

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

$$
\begin{array}{rcl}
M & ::= & x \\
& | & \lambda x : \sigma . M \\
& | & M\,N \\
& | & unit \\
& | & ref\ \ M \\
& | & !M \\
& | & M := N \\
& | & l \\
& | & \ldots
\end{array}
$$

Juicios de tipado

$$\Gamma \triangleright I : \textcolor{red}{\textbf{?}}$$

Juicios de tipado

$$\Gamma \rhd l : \mathbf{?}$$

- 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

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

Reglas de tipado - Definitivas

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

$$\frac{\Gamma|\Sigma \rhd M_1 : \mathit{Ref}\ \ \sigma}{\Gamma|\Sigma \rhd !M_1 : \sigma}\ (\text{T-DeRef})$$

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

$$\frac{\Sigma(l) = \sigma}{\Gamma|\Sigma \rhd l : \mathit{Ref}\ \ \sigma}\ (\text{T-Loc})$$

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.

$$M \mid \mu \to M' \mid \mu'$$

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

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

Juicios de evaluación en un paso

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

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

Juicios de evaluación en un paso

$$\frac{M_1 \mid \mu \to M_1' \mid \mu'}{M_1 := M_2 \mid \mu \to M_1' := M_2 \mid \mu'} \text{(E-Assign1)}$$

$$\frac{M_2 \mid \mu \to M_2' \mid \mu'}{V := M_2 \mid \mu \to V := M_2' \mid \mu'} \text{(E-Assign2)}$$

$$\frac{}{l := V \mid \mu \to unit \mid \mu[l \mapsto V]} \text{(E-Assign)}$$

Juicios de evaluación en un paso

$$\frac{M_1 \mid \mu \to M_1' \mid \mu'}{\mathit{ref}\ \ M_1 \mid \mu \to \mathit{ref}\ \ M_1' \mid \mu'} \text{ (E-Ref)}$$

$$\frac{l \notin \mathit{Dom}(\mu)}{\mathit{ref}\ \ V \mid \mu \to l \mid \mu \oplus (l \mapsto V)} \text{ (E-RefV)}$$

Revisitar los juicios de evaluación para los restantes términos

$$\frac{M_1 \mid \mu \to M_1' \mid \mu'}{M_1\, M_2 \mid \mu \to M_1'\, M_2 \mid \mu'} \text{ (E-App1)}$$

$$\frac{M_2 \mid \mu \to M_2' \mid \mu'}{V_1\, M_2 \mid \mu \to V_1\, M_2' \mid \mu'} \text{ (E-App2)}$$

$$\frac{}{(\lambda x : \sigma.M)\, V \mid \mu \to M\{x \leftarrow V\} \mid \mu} \text{ (E-AppAbs)}$$

Las de los restantes tipos (*Nat*, *Bool*, ...) son similares.
Nota: Estas reglas no modifican el store.

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 $M \to M'$

Preservación

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

Debemos reformular estos resultados en el marco de referencias.

Preservación - Formulación ingenua

La formulación ingenua siguiente es errónea:

$$\Gamma|\Sigma \rhd M : \sigma \quad \text{y} \quad M \mid \mu \to M' \mid \mu' \quad \text{implica} \quad \Gamma|\Sigma \rhd M' : \sigma$$

- 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

$$\Gamma|\Sigma \rhd M : \sigma \quad \text{y} \quad M \mid \mu \to M' \mid \mu' \quad \text{implica} \quad \Gamma|\Sigma \rhd M' : \sigma$$

Preservación - Formulación ingenua

$$\Gamma|\Sigma \rhd M : \sigma \quad \text{y} \quad M \mid \mu \to M' \mid \mu' \quad \text{implica} \quad \Gamma|\Sigma \rhd M' : \sigma$$

Supongamos que

- $M = !l$
- $\Gamma = \emptyset$
- $\Sigma(l) = Nat$
- $\mu(l) = true$

Observar que

- $\Gamma|\Sigma \rhd M : Nat$ y
- $M \mid \mu \to true \mid \mu$
- pero $\Gamma|\Sigma \rhd true : Nat$ no vale.

Preservación - Formulación ingenua

$$\Gamma|\Sigma \rhd M : \sigma \quad \text{y} \quad M \,|\, \mu \to M' \,|\, \mu' \quad \text{implica} \quad \Gamma|\Sigma \rhd M' : \sigma$$

Supongamos que

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

Observar que

- $\Gamma|\Sigma \rhd M : Nat$ y
- $M \,|\, \mu \to true \,|\, \mu$
- pero $\Gamma|\Sigma \rhd true : Nat$ 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.

$$\text{Si} \quad \Gamma|\Sigma \rhd M : \sigma \quad \text{y} \quad M \,|\, \mu \to N \,|\, \mu' \quad \text{y} \quad \Gamma|\Sigma \rhd \mu,$$

$$\text{entonces} \quad \Gamma|\boxed{\Sigma} \rhd N : \sigma.$$

- Esto es casi correcto.

Preservación - Reformulada

Reformulamos preservación del siguiente modo.

$$\text{Si} \quad \Gamma|\Sigma \triangleright M : \sigma \quad \text{y} \quad M \,|\, \mu \to N \,|\, \mu' \quad \text{y} \quad \Gamma|\Sigma \triangleright \mu,$$

$$\text{entonces} \quad \Gamma|\boxed{\Sigma} \triangleright N : \sigma.$$

- 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

- $\Gamma|\Sigma \triangleright M : \sigma$
- $M \mid \mu \to N \mid \mu'$
- $\Gamma|\Sigma \triangleright \mu$

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. $\emptyset|\Sigma \triangleright M : \sigma$ para algún Σ, σ) entonces:

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

Ejemplo

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

$$\rightarrow$$

Ejemplo

$$
\begin{array}{ll}
 & \mathit{let}\ x = \mathit{ref}\ \underline{2}\ \mathit{in}\ (\lambda\_ : \mathit{Unit}.!x)\,(x := \mathit{succ}(!x)) \mid \mu \\
\rightarrow & \mathit{let}\ x = l_1\ \mathit{in}\ (\lambda\_ : \mathit{Unit}.!x)\,(x := \mathit{succ}(!x)) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\rightarrow &
\end{array}
$$

Ejemplo

$$
\begin{array}{ll}
& let\ x = ref\ \underline{2}\ in\ (\lambda\_ : Unit.!x)\,(x := succ(!x)) \mid \mu \\
\rightarrow & let\ x = l_1\ in\ (\lambda\_ : Unit.!x)\,(x := succ(!x)) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\rightarrow & (\lambda\_ : Unit.!l_1)\,(l_1 := succ(!l_1)) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\rightarrow &
\end{array}
$$

Ejemplo

$$
\begin{array}{rl}
 & let\ x = ref\ \underline{2}\ in\ (\lambda_{\_} : Unit.!x)\,(x := succ(!x)) \mid \mu \\
\to & let\ x = l_1\ in\ (\lambda_{\_} : Unit.!x)\,(x := succ(!x)) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\to & (\lambda_{\_} : Unit.!l_1)\,(l_1 := succ(!l_1)) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\to & (\lambda_{\_} : Unit.!l_1)\,(l_1 := succ(\underline{2})) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\to &
\end{array}
$$

Ejemplo

$$
\begin{array}{rl}
 & let\ x = ref\ \underline{2}\ in\ (\lambda\_ : Unit.!x)\,(x := succ(!x)) \mid \mu \\
\to & let\ x = l_1\ in\ (\lambda\_ : Unit.!x)\,(x := succ(!x)) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\to & (\lambda\_ : Unit.!l_1)\,(l_1 := succ(!l_1)) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\to & (\lambda\_ : Unit.!l_1)\,(l_1 := succ(\underline{2})) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\to & (\lambda\_ : Unit.!l_1)\ unit \mid (\mu \oplus (l_1 \mapsto \underline{2}))[l_1 \mapsto \underline{3}] \\
\to &
\end{array}
$$

Ejemplo

$$
\begin{array}{ll}
& \mathit{let}\ x = \mathit{ref}\ \underline{2}\ \mathit{in}\ (\lambda\_ : \mathit{Unit}.!x)\,(x := \mathit{succ}(!x)) \mid \mu \\
\rightarrow & \mathit{let}\ x = l_1\ \mathit{in}\ (\lambda\_ : \mathit{Unit}.!x)\,(x := \mathit{succ}(!x)) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\rightarrow & (\lambda\_ : \mathit{Unit}.!l_1)\,(l_1 := \mathit{succ}(!l_1)) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\rightarrow & (\lambda\_ : \mathit{Unit}.!l_1)\,(l_1 := \mathit{succ}(\underline{2})) \mid \mu \oplus (l_1 \mapsto \underline{2}) \\
\rightarrow & (\lambda\_ : \mathit{Unit}.!l_1)\ \mathit{unit} \mid (\mu \oplus (l_1 \mapsto \underline{2}))[l_1 \mapsto \underline{3}] \\
\rightarrow & !l_1 \mid \mu \oplus (l_1 \mapsto \underline{3}) \\
\rightarrow & \underline{3} \mid \mu \oplus (l_1 \mapsto \underline{3})
\end{array}
$$

Ejemplo

Sea

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

$$
\begin{array}{ll}
 & M\,(ref\ \ (\lambda x : Unit.x)) \mid \mu \\
\to &
\end{array}
$$

Ejemplo

Sea

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

$$\begin{array}{ll} & M\,(ref\ (\lambda x : Unit.x)) \mid \mu \\ \to & M\ l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\ \to & \end{array}$$

Ejemplo

Sea

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

$$\begin{array}{ll} & M\,(ref\ (\lambda x : Unit.x)) \mid \mu \\ \to & M\ l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\ \to & let\ f = !l_1\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\ \to & \end{array}$$

Ejemplo

Sea

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

$$\begin{array}{rl} & M\,(ref\ (\lambda x : Unit.x)) \mid \mu \\ \to & M\ l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\ \to & let\ f = !l_1\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\ \to & let\ f = \lambda x : Unit.x\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\ \to & \end{array}$$

Ejemplo

Sea

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

$$
\begin{array}{rl}
 & M\,(ref\ (\lambda x : Unit.x)) \mid \mu \\
\to & M\, l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\
\to & let\ f = !l_1\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\
\to & let\ f = \lambda x : Unit.x\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\
\to & (l_1 := \lambda x : Unit.(\lambda x : Unit.x)\, x); (!l_1)\ unit \mid \ldots \\
\to &
\end{array}
$$

Ejemplo

Sea

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

$$\begin{array}{ll} & M\,(ref\ (\lambda x : Unit.x)) \mid \mu \\ \to & M\ l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\ \to & let\ f = !l_1\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\ \to & let\ f = \lambda x : Unit.x\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\ \to & (l_1 := \lambda x : Unit.(\lambda x : Unit.x)\ x); (!l_1)\ unit \mid \ldots \\ \to & unit; (!l_1)\ unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(\lambda x : Unit.x)\ x) \\ \to & \end{array}$$

Ejemplo

Sea

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

$$\begin{array}{rl} & M\,(ref\ (\lambda x : Unit.x)) \mid \mu \\ \to & M\ l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\ \to & let\ f = !l_1\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\ \to & let\ f = \lambda x : Unit.x\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\ \to & (l_1 := \lambda x : Unit.(\lambda x : Unit.x)\ x); (!l_1)\ unit \mid \ldots \\ \to & unit; (!l_1)\ unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(\lambda x : Unit.x)\ x) \\ \to & (!l_1)\ unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(\lambda x : Unit.x)\ x) \\ \to & \end{array}$$

Ejemplo

Sea

$$M = \lambda r : Ref(Unit \to Unit).\; let\ f = !r\ in\ (r := \lambda x : Unit.f\,x); (!r)\,unit$$

$$\begin{array}{ll}
 & M\,(ref\ (\lambda x : Unit.x)) \mid \mu \\
\to & M\,l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\
\to & let\ f = !l_1\ in\ (l_1 := \lambda x : Unit.f\,x); (!l_1)\,unit \mid \ldots \\
\to & let\ f = \lambda x : Unit.x\ in\ (l_1 := \lambda x : Unit.f\,x); (!l_1)\,unit \mid \ldots \\
\to & (l_1 := \lambda x : Unit.(\lambda x : Unit.x)\,x); (!l_1)\,unit \mid \ldots \\
\to & unit; (!l_1)\,unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(\lambda x : Unit.x)\,x) \\
\to & (!l_1)\,unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(\lambda x : Unit.x)\,x) \\
\to & (\lambda x : Unit.(\lambda x : Unit.x)\,x)\,unit \mid \ldots \\
\to &
\end{array}$$

Ejemplo

Sea

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

$$\begin{array}{rl} & M\,(ref\ (\lambda x : Unit.x)) \mid \mu \\ \to & M\ l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\ \to & let\ f = !l_1\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\ \to & let\ f = \lambda x : Unit.x\ in\ (l_1 := \lambda x : Unit.f\ x); (!l_1)\ unit \mid \ldots \\ \to & (l_1 := \lambda x : Unit.(\lambda x : Unit.x)\ x); (!l_1)\ unit \mid \ldots \\ \to & unit; (!l_1)\ unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(\lambda x : Unit.x)\ x) \\ \to & (!l_1)\ unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(\lambda x : Unit.x)\ x) \\ \to & (\lambda x : Unit.(\lambda x : Unit.x)\ x)\ unit \mid \ldots \\ \to & (\lambda x : Unit.x)\ unit \mid \ldots \\ \to & unit \mid \ldots \end{array}$$

Ejemplo

Sea

$$\begin{array}{rcl} M & = & \lambda r : Ref(Unit \to Unit). \\ & & \quad \textit{let } f = !r \\ & & \quad \textit{in } (r := \lambda x : Unit.f\ x); (!r)\ unit \end{array}$$

Reemplazamos f por $!r$ y nos queda

$$\begin{array}{rcl} M' & = & \lambda r : Ref\ (Unit \to Unit). \\ & & \quad (r := \lambda x : Unit.(!r)\ x); (!r)\ unit \end{array}$$

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

Ejemplo

$$\begin{array}{rcl} M' & = & \lambda r : Ref\ (Unit \to Unit). \\ & & \quad (r := \lambda x : Unit.(!r)\, x); (!r)\, unit \end{array}$$

$$M'\, (ref\ (\lambda x : Unit.x)) \mid \mu$$

$$\to$$

Ejemplo

$$
\begin{array}{rcl}
M' & = & \lambda r : \mathit{Ref}\ (\mathit{Unit} \to \mathit{Unit}). \\
 & & \quad (r := \lambda x : \mathit{Unit}.(!r)\, x); (!r)\, \mathit{unit}
\end{array}
$$

$$
\begin{array}{ll}
 & M'\, (\mathit{ref}\ (\lambda x : \mathit{Unit}.x)) \mid \mu \\
\to & M'\, l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : \mathit{Unit}.x) \\
\to &
\end{array}
$$

Ejemplo

$$\begin{array}{rcl} M' & = & \lambda r : Ref\ (Unit \to Unit). \\ & & \quad (r := \lambda x : Unit.(!r)\, x); (!r)\, unit \end{array}$$

$$\begin{array}{rl} & M'\, (ref\ (\lambda x : Unit.x)) \mid \mu \\ \to & M'\, l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\ \to & (l_1 := \lambda x : Unit.(!l_1)\, x); (!l_1)\, unit \mid \ldots \\ \to & \end{array}$$

Ejemplo

$$M' = \lambda r : Ref\ (Unit \to Unit).$$
$$(r := \lambda x : Unit.(!r)\,x); (!r)\,unit$$

$$
\begin{array}{ll}
 & M'\,(ref\ (\lambda x : Unit.x)) \mid \mu \\
\to & M'\,l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\
\to & (l_1 := \lambda x : Unit.(!l_1)\,x); (!l_1)\,unit \mid \ldots \\
\to & unit; (!l_1)\,unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(!l_1)\,x) \\
\to &
\end{array}
$$

Ejemplo

$$
\begin{array}{rcl}
M' & = & \lambda r : \mathit{Ref}\ (\mathit{Unit} \to \mathit{Unit}). \\
& & \quad (r := \lambda x : \mathit{Unit}.(!r)\, x); (!r)\, \mathit{unit}
\end{array}
$$

$$
\begin{array}{rl}
& M'\, (\mathit{ref}\ (\lambda x : \mathit{Unit}.x)) \mid \mu \\
\to & M'\, l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : \mathit{Unit}.x) \\
\to & (l_1 := \lambda x : \mathit{Unit}.(!l_1)\, x); (!l_1)\, \mathit{unit} \mid \ldots \\
\to & \mathit{unit}; (!l_1)\, \mathit{unit} \mid \mu \oplus (l_1 \mapsto \lambda x : \mathit{Unit}.(!l_1)\, x) \\
\to & \boxed{(!l_1)\, \mathit{unit}} \mid \ldots \\
\to &
\end{array}
$$

Ejemplo

$$M' \;=\; \lambda r : Ref\ (Unit \to Unit).\\ (r := \lambda x : Unit.(!r)\, x); (!r)\ unit$$

$$\begin{array}{ll}
 & M'\,(ref\ (\lambda x : Unit.x)) \mid \mu \\
\to & M'\ l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\
\to & (l_1 := \lambda x : Unit.(!l_1)\, x); (!l_1)\ unit \mid \ldots \\
\to & unit; (!l_1)\ unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(!l_1)\, x) \\
\to & \boxed{(!l_1)\ unit} \mid \ldots \\
\to & (\lambda x : Unit.(!l_1)\, x)\ unit \mid \ldots \\
\to &
\end{array}$$

Ejemplo

$$\begin{array}{rcl} M' & = & \lambda r : Ref\ (Unit \to Unit). \\ & & \quad (r := \lambda x : Unit.(!r)\,x); (!r)\,unit \end{array}$$

$$\begin{array}{rl} & M'\,(ref\ (\lambda x : Unit.x)) \mid \mu \\ \to & M'\,l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\ \to & (l_1 := \lambda x : Unit.(!l_1)\,x); (!l_1)\,unit \mid \ldots \\ \to & unit; (!l_1)\,unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(!l_1)\,x) \\ \to & \boxed{(!l_1)\,unit} \mid \ldots \\ \to & (\lambda x : Unit.(!l_1)\,x)\,unit \mid \ldots \\ \to & \boxed{(!l_1)\,unit} \mid \ldots \\ \to & \ldots \end{array}$$

Ejemplo

$$
\begin{array}{rcl}
M' & = & \lambda r : Ref\ (Unit \to Unit). \\
 & & \quad (r := \lambda x : Unit.(!r)\,x); (!r)\,unit
\end{array}
$$

$$
\begin{array}{ll}
 & M'\,(ref\ (\lambda x : Unit.x)) \mid \mu \\
\to & M'\,l_1 \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.x) \\
\to & (l_1 := \lambda x : Unit.(!l_1)\,x); (!l_1)\,unit \mid \ldots \\
\to & unit; (!l_1)\,unit \mid \mu \oplus (l_1 \mapsto \lambda x : Unit.(!l_1)\,x) \\
\to & \boxed{(!l_1)\,unit} \mid \ldots \\
\to & (\lambda x : Unit.(!l_1)\,x)\,unit \mid \ldots \\
\to & \boxed{(!l_1)\,unit} \mid \ldots \\
\to & \ldots
\end{array}
$$

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

$$M \quad ::= \quad \ldots \mid \mathsf{fix}\ \ M$$

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

$$\frac{\Gamma \rhd M : \sigma_1 \to \sigma_1}{\Gamma \rhd \mathsf{fix}\ \ M : \sigma_1}\,(\textrm{T-}\textsc{Fix})$$

Semántica operacional small-step

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

$$\frac{M_1 \to M_1'}{\mathsf{fix}\ \ M_1 \to \mathsf{fix}\ \ M_1'}\ (\textrm{E-}\textsc{Fix})$$

$$\frac{}{\mathsf{fix}\ (\lambda x : \sigma.M) \to M\{x \leftarrow \mathsf{fix}\ (\lambda x : \sigma.M)\}}\ (\textrm{E-}\textsc{FixBeta})$$

Ejemplos

Sea M el término

$$\begin{array}{l}\lambda f : Nat \to Nat. \\ \;\lambda x : Nat. \\ \;\; \mathit{if}\ \ \text{iszero}(x)\ \mathit{then}\ \underline{1}\ \mathit{else}\ x * f(\text{pred}(x))\end{array}$$

en

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

Ejemplos

Ahora podemos definir funciones parciales:

Ejemplos

Ahora podemos definir funciones parciales:

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

Ejemplos

Sea M el término

$$\begin{array}{l}\lambda s : Nat \to Nat \to Nat. \\ \ \lambda x : Nat. \\ \ \ \lambda y : Nat. \\ \ \ \ \mathit{if}\ \ \mathrm{iszero}(x)\ \mathit{then}\ y\ \mathit{else}\ \ \mathrm{succ}(s\ \mathrm{pred}(x)\ y)\end{array}$$

en

$$\mathit{let}\ \mathit{suma} = \mathrm{fix}\ M\ \mathit{in}\ \mathit{suma}\,\underline{2}\,\underline{3}$$

Letrec

Una construcción alternativa para definir funciones recursivas es

$$letrec\ f : \sigma \to \sigma = \lambda x : \sigma.M\ in\ N$$

Por ejemplo,

$$\begin{array}{l} letrec \\ \ fact : Nat \to Nat = \\ \lambda x : Nat.if\ \ iszero(x)\ then\ \underline{1}\ else\ x * fact(pred(x)) \\ \ \ in\ fact\,\underline{3} \end{array}$$

letrec puede escribirse en términos de fix del siguiente modo:

$$let\ f = \mathrm{fix}\,(\lambda f : \sigma \to \sigma.\lambda x : \sigma.M)\ in\ N$$