Lambda Cálculo Tipado (2/3)

Eduardo Bonelli

Departamento de Computación, FCEyN, UBA

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

Alonzo Church, 1932

6 de septiembre de 2012

La clase pasada

- Lambda cálculo 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

Corrección = Progreso + 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 $\Gamma \rhd M : \sigma$ y $M \to N$, entonces $\Gamma \rhd N : \sigma$

La evaluación preserva tipos

Hoy - Dos extensiones más

- Referencias

  Programación Imperativa $=$ Programación 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 $let\ x = \underline{2}\ in\ M$
  - 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 Lambda Cálculo Tipado con variables mutables

Operaciones básicas

Alocación

ref M genera una referencia fresca cuyo contenido es el valor de M

Dereferencia

$!x$ dereferencia la referencia x y retorna su contenido

Asignación

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

Ejemplos

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

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
  - Carece de sentido 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

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

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

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

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

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

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

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

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 item del primer bullet puede escribirse también

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

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 modelizar 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:

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

Los valores posibles ahora incluyen las direcciones

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

Dado que los valores son un subconjunto de los términos,

- debemos ampliar los términos con direcciones
- estas son producto de la formalización y no se pretende que sean utilizadas por el programador

Términos extendidos

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

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

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

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

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

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

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

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

Juicios de evaluación en un paso (1/4)

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

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

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

Nota: Estas reglas no modifican el store

Juicios de evaluación en un paso (2/4)

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

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

Juicios de evaluación en un paso (3/4)

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

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

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

Juicios de evaluación en un paso (4/4)

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

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

Ejemplo

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

Ejemplo

Sea

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

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

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 y ver qué pasa...

Ejemplo

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

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

Nota: No todo término bien tipado termina en LC con 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 $M \to M'$

Preservación

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

Debemos reformular estos resultados en el marco de referencias

Preservación - Formulación ingenua

La formulación ingenua siguiente es errónea:

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

- 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

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

Supongamos que

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

Observar que

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

Preservación - Formulación ingenua

Formulación ingenua

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

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

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

- Esto es casi correcto
- No contempla la posibilidad de que el Σ encuadrado haya crecido en dominio respecto a Σ
  - Por posibles alocaciones

Preservación - Definitiva

- $\Gamma|\Sigma \triangleright M : \sigma$,
- $M \to N$ y
- $\Gamma|\Sigma \triangleright \mu : \Sigma$

implica que existe $\Sigma' \supseteq \Sigma$ tal que $\Gamma|\Sigma' \triangleright N : \sigma$

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 $M \,|\, \mu \rightarrow M' \,|\mu'$

Recursión

Ecuación recursiva

$$f = \ldots f \ldots f \ldots$$

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

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

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

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

Semántica operacional small-step

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

$$\frac{M_1 \to M_1'}{\mathit{fix}\ M_1 \to \mathit{fix}\ M_1'}\ (\text{E-}\textsc{Fix})$$

$$\frac{}{\mathit{fix}\ (\lambda x : \sigma.M) \to M_2\{x \leftarrow \mathit{fix}\ (\lambda x : \sigma.M)\}}\ (\text{E-}\textsc{FixBeta})$$

Ejemplos

Sea M el término

$$\begin{array}{l}\lambda f : Nat \to Nat. \\ \ \lambda x : Nat. \\ \ \ if\ \ x = 0\ then\ \underline{1}\ else\ x * f(pred(x))\end{array}$$

en

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

Ejemplos

$$\mathit{fix}\,(\lambda x : \mathit{Nat}.x + 1)$$

Ejemplos

Sea M el término

$$
\begin{array}{l}
\lambda s : Nat \to Nat \to Nat. \\
\ \lambda x : Nat. \\
\ \ \lambda y : Nat. \\
\ \ \ \mathit{if}\ x = 0\ \mathit{then}\ y\ \mathit{else}\ \mathit{succ}(s\, \mathit{pred}(x)\, y)
\end{array}
$$

en

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

Letrec

Una construcción alternativa para definir funciones recursivas es

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

Por ejemplo,

$$\begin{array}{l} letrec \\ \quad fact : Nat \to Nat = \lambda x : Nat.if\ x = 0\ then\ \underline{1}\ else\ x * f(pred(x)) \\ \quad\ in\ fact\,\underline{3} \end{array}$$

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

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