Programación Funcional Avanzada

Alberto Pardo   Marcos Viera

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

Cálculo Lambda Tipado

Cálculo Lambda sin tipos

Lenguaje de términos

$$\begin{array}{rl} t ::= & x \\ \mid & \lambda x.\ t \\ \mid & t\ t \end{array}$$

Valores:

$$v ::= \lambda x.\ t$$

$\lambda^{\rightarrow}$ – Cálculo Lambda Simplemente Tipado

- **Términos:**

$$
\begin{aligned}
t ::= {} & x \\
\mid {} & \lambda x^{\tau} .\, t \\
\mid {} & t \; t
\end{aligned}
$$

En las abstracciones las variables ahora tienen anotado el tipo.

- **Tipos:**

$$\tau ::= b \mid \tau \rightarrow \tau$$

$b \in B$ son constantes de tipo como *bool*, *int*, *char*, *float*, etc.

Juicio de Tipado

- **Contexto de tipado:** conjunto de hipótesis de tipado sobre variables

  $$\Gamma = \{ x_1 : \tau_1, ..., x_n : \tau_n \}$$

  tal que ninguna variable x_i aparece dos veces.

  Escribiremos $\Gamma, x : \tau$ en lugar de $\Gamma \cup \{ x : \tau \}$.

- El siguiente juicio establece que en el ambiente Γ, el término t tiene tipo τ.

  $$\Gamma \vdash t : \tau$$

$\lambda^{\rightarrow}$ – Reglas de tipado

$$\Gamma, x : \tau \vdash x : \tau \ (\mathit{var}) \qquad \frac{\Gamma, x : \tau \vdash t : \tau'}{\Gamma \vdash \lambda x^{\tau}.t : \tau \rightarrow \tau'} \ (\mathit{abs})$$

$$\frac{\Gamma \vdash t : \tau \rightarrow \tau' \qquad \Gamma \vdash t' : \tau}{\Gamma \vdash t\ t' : \tau'} \ (\mathit{app})$$

Ejemplo: $\mathit{suc} : \mathit{int} \rightarrow \mathit{int} \vdash \lambda x^{\mathit{int}}.\ \mathit{suc}\ x : \mathit{int} \rightarrow \mathit{int}$

Propiedades de $\lambda^{\rightarrow}$

Unicidad de Tipos: En un contexto Γ dado, un término t (con todas sus variables libres en Γ) tiene como máximo un tipo.

Progreso: Si t es un término cerrado bien-tipado ($\vdash t : \tau$), entonces o bien t es un valor o existe algún t' tal que $t \leadsto t'$.

Preservación de tipos bajo reducción: Si $\Gamma \vdash t : \tau$ y $t \leadsto t'$, entonces $\Gamma \vdash t' : \tau$.

Recursión General

- No se pueden definir computaciones divergentes en $\lambda^{\rightarrow}$.
- El Combinador de Punto Fijo no se puede definir en $\lambda^{\rightarrow}$.
- Para poder tener recursión uno debe agregar un combinador *fix* (uno para cada tipo τ) como primitiva del lenguaje:

$$\begin{aligned} t ::= {} & x \\ \mid {} & \lambda x^{\tau}.\, t \\ \mid {} & t\ t \\ \mid {} & \mathit{fix}_{\tau}\ t \end{aligned}$$

tal que

$$\frac{\Gamma \vdash t : \tau \rightarrow \tau}{\Gamma \vdash \mathit{fix}_{\tau}\ t : \tau}(\mathit{fix}) \qquad \text{y} \qquad \mathit{fix}_{\tau}\ t \rightarrow_{\beta} t(\mathit{fix}_{\tau}\ t)$$

System F – Cálculo Lambda Polimórfico

- Términos

$$
\begin{array}{rll}
t ::= & x & \\
| & \lambda x^{\tau}.\ t & \\
| & t\ t & \\
| & \Lambda\ \alpha.\ t & \text{-- abstracción de tipo} \\
| & t\ \tau & \text{-- aplicación de tipo}
\end{array}
$$

- Reglas reducción (β):

$$(\lambda x^{\tau}.\ t)\ t' \rightarrow t\ [t' \,/\, x]$$
$$(\Lambda\ \alpha.\ t)\ \tau\ \rightarrow t\ [\tau \,/\, \alpha]$$

Tipado en System F

- Tipos:

$$
\begin{array}{rll}
\tau ::= & \alpha & \text{-- variables de tipo} \\
\mid & \tau \to \tau & \text{-- tipo funcional} \\
\mid & \forall\, \alpha.\, \tau & \text{-- tipo universal}
\end{array}
$$

- Contextos (ahora son listas de hipótesis y no conjuntos)

$$
\begin{array}{rll}
\Gamma ::= & \Phi & \text{-- contexto vacío} \\
\mid & \Gamma, x : \tau & \text{-- hipótesis de tipado de variables} \\
\mid & \Gamma, \alpha & \text{-- hipótesis de tipo}
\end{array}
$$

- Juicio de tipado:

$$\Gamma \vdash t : \tau$$

Reglas de Tipado

Mismas reglas para variables, abstracción y aplicación:

$$\Gamma, x : \tau \vdash x : \tau \ (var) \qquad \frac{\Gamma, x : \tau \vdash t : \tau'}{\Gamma \vdash \lambda x^{\tau}.t : \tau \to \tau'}(abs)$$

$$\frac{\Gamma \vdash t : \tau \to \tau' \qquad \Gamma \vdash t' : \tau}{\Gamma \vdash t\ t' : \tau'}(app)$$

Nuevas reglas para abstracción y aplicación de tipos:

$$\frac{\Gamma, \alpha \vdash t : \tau}{\Gamma \vdash \Lambda\alpha.t : \forall\alpha.\tau}(typeabs) \qquad \frac{\Gamma \vdash t : \forall\alpha.\tau}{\Gamma \vdash t\ \tau' : \tau[\tau'/\alpha]}(typeapp)$$

Ejemplo: función identidad

Identidad polimórfica:

$$id = \Lambda\, \alpha.\, \lambda x^{\alpha}.\, x : \forall\, \alpha.\, \alpha \to \alpha$$

Identidad sobre los naturales:

$$idnat = id\ Nat : Nat \to Nat$$

Identidad polimórfica aplicada a sí misma:

$$id\ (\forall\, \alpha.\, \alpha \to \alpha)\ id : \forall\, \alpha.\, \alpha \to \alpha$$

Representación de tipos en System F

Codificación de los booleanos:

$$Bool = \forall\, \alpha.\, \alpha \rightarrow \alpha \rightarrow \alpha$$

Smart constructors para true y false:

$$T : Bool$$
$$T = \Lambda\, \alpha.\, \lambda t^{\alpha}.\, \lambda f^{\alpha}.\, t^{\alpha}$$

$$F : Bool$$
$$F = \Lambda\, \alpha.\, \lambda t^{\alpha}.\, \lambda f^{\alpha}.\, f^{\alpha}$$

Representación de Tipos: booleanos (2)

Algunas funciones sobre los booleanos:

$$IF\_THEN\_ELSE : \forall\ \alpha.\ Bool \to \alpha \to \alpha \to \alpha$$
$$IF\_THEN\_ELSE = \Lambda\ \alpha.\ \lambda b^{Bool}.\ \lambda t^{\alpha}.\ \lambda e^{\alpha}.\ b\ \alpha\ t\ e$$

$$NOT\ :\ Bool \to Bool$$
$$NOT = \lambda b^{Bool}.\ \Lambda\ \alpha.\ b\ \alpha\ F\ T$$

$$AND\ :\ Bool \to Bool \to Bool$$
$$AND = \lambda a.\ \lambda b.\ a\ b\ \ F$$

$$OR\ :\ Bool \to Bool \to Bool$$
$$OR = \lambda a.\ \lambda b.\ a\ T\ b$$

Representación de Tipos: pares

Codificación del tipo par (a, b):

$$Pair\ a\ b = \forall\ \alpha.\ (a \to b \to \alpha) \to \alpha$$

Smart constructor:

$$\begin{aligned} PAIR &: a \to b \to Pair\ a\ b \\ PAIR &= \lambda x^a.\ \lambda y^b.\ \Lambda\ \alpha.\ \lambda p^{a \to b \to \alpha}.\ p\ x\ y \end{aligned}$$

Proyecciones:

$$\begin{aligned} FST &: Pair\ a\ b \to a \\ FST &= \lambda p^{Pair\ a\ b}.\ p\ a\ (\lambda x^a.\ \lambda y^b.\ x^a) \end{aligned}$$

$$\begin{aligned} SND &: Pair\ a\ b \to b \\ SND &= \lambda p^{Pair\ a\ b}.\ p\ b\ (\lambda x^a.\ \lambda y^b.\ y^b) \end{aligned}$$

Church numerals

Los números naturales se definen mediante dos constructores:

- $S : Nat \to Nat$ (sucesor)
- $Z : Nat$ (cero)

Codificación del tipo de los naturales:

$$Nat = \forall\, \alpha.\ (\alpha \to \alpha) \to \alpha \to \alpha$$

Cada natural n es codificado por un término lambda $\overline{n}$ de tipo Nat:

$$\overline{n} = \Lambda\, \alpha.\ \lambda s^{\alpha \to \alpha}.\ \lambda z^{\alpha}.\ s^{n}\ z$$

Ejemplos,

$$\begin{aligned}
\overline{0} &= \Lambda\, \alpha.\ \lambda s^{\alpha \to \alpha}.\ \lambda z^{\alpha}.\ z \\
\overline{1} &= \Lambda\, \alpha.\ \lambda s^{\alpha \to \alpha}.\ \lambda z^{\alpha}.\ s\ z \\
\overline{2} &= \Lambda\, \alpha.\ \lambda s^{\alpha \to \alpha}.\ \lambda z^{\alpha}.\ s\ (s\ z)
\end{aligned}$$

Church numerals: smart constructors

Definimos dos funciones que corresponden a los constructores:

$$Z \ : \ Nat$$
$$Z = \Lambda\, \alpha.\ \lambda s^{\alpha \to \alpha}.\ \lambda z^{\alpha}.\ z$$

$$S \quad : \ Nat \to Nat$$
$$S\ n = \Lambda\, \alpha.\ \lambda s^{\alpha \to \alpha}.\ \lambda z^{\alpha}.\ s\ (n\ \alpha\ s\ z)$$

Notar que $\overline{n} = S^n\ Z$.

Church numerals: iterador

Iterador (operador *fold*) para a los naturales:

$$foldN : \forall\, \tau.\ (\tau \to \tau) \to \tau \to Nat \to \tau$$
$$foldN\ \tau\ h\ v\ n = n\ \tau\ h\ v$$

que se comporta de esta forma:

$$\begin{array}{lll} foldN\ \tau\ h\ v\ Z & \to^* & v \\ foldN\ \tau\ h\ v\ (S\ \overline{n}) & \to^* & h^{n+1}\ v \end{array}$$

Church numerals: funciones sobre naturales

Suma de dos naturales en términos de *foldN*:

$$\begin{array}{l} PLUS \;:\; Nat \to Nat \to Nat \\ PLUS = \lambda m^{Nat}.\, \lambda n^{Nat}.\, foldN\ Nat\ S\ m\ n \end{array}$$

La función *PLUS* se comporta de esta manera:

$$\begin{array}{lll} PLUS\ m\ Z & \to^{*} & m \\ PLUS\ m\ (S\ \overline{n}) & \to^{*} & S^{n+1}\ m \end{array}$$

Test de si un natural es cero:

$$\begin{array}{l} ISZERO : Nat \to Bool \\ ISZERO = \lambda n^{Nat}.\, foldN\ Bool\ (\lambda x^{Bool}.\, F)\ T\ n \end{array}$$