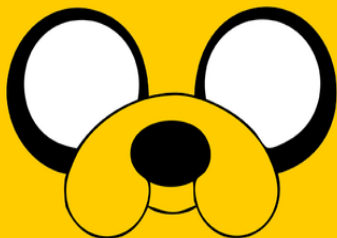


Polimorfismo



Mauro Jaskelioff

29/09/17

Repaso

- Recordemos la sintaxis abstracta de nuestro lenguaje de expresiones aritméticas y funciones:

$$\begin{aligned} t ::= & \text{true} \mid \text{false} \mid \text{if } t \text{ then } t \text{ else } t \\ & \mid 0 \mid \text{succ } t \mid \text{pred } t \mid \text{iszero } t \\ & \mid x \mid \lambda x: T. t \mid t t \end{aligned}$$

- Los valores son:

$$\begin{aligned} v ::= & \text{true} \mid \text{false} \mid nv \mid \lambda x: T. t \\ nv ::= & 0 \mid \text{succ } nv \end{aligned}$$

- Los tipos son

$$T ::= \text{Bool} \mid \text{Nat} \mid T \rightarrow T$$

Reglas de Tipado

$$\Gamma \vdash \text{true} : \text{Bool} \quad (\text{T-TRUE})$$
$$\Gamma \vdash \text{false} : \text{Bool} \quad (\text{T-FALSE})$$
$$\frac{\Gamma \vdash t_1 : \text{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : T} \quad (\text{T-IF})$$
$$\Gamma \vdash 0 : \text{Nat} \quad (\text{T-ZERO})$$
$$\frac{\Gamma \vdash t_1 : \text{Nat}}{\Gamma \vdash \text{succ } t_1 : \text{Nat}} \quad (\text{T-SUCC})$$
$$\frac{\Gamma \vdash t_1 : \text{Nat}}{\Gamma \vdash \text{pred } t_1 : \text{Nat}} \quad (\text{T-PRED})$$

Reglas de Tipado (cont.)

:

$$\frac{\Gamma \vdash t_1 : \text{Nat}}{\Gamma \vdash \text{iszero } t_1 : \text{Bool}} \quad (\text{T-ISZERO})$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \quad (\text{T-VAR})$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1. t_2 : T_1 \rightarrow T_2} \quad (\text{T-ABS})$$

$$\frac{\Gamma \vdash t_2 : T_1 \rightarrow T_2 \quad \Gamma \vdash t_1 : T_1}{\Gamma \vdash t_2 \ t_1 : T_2} \quad (\text{T-APP})$$

Motivación

- Consideremos las funciones

$\text{doubleNat} = \lambda f: \text{Nat} \rightarrow \text{Nat}. \lambda x: \text{Nat}. f (f x)$

$\text{doubleBool} = \lambda f: \text{Bool} \rightarrow \text{Bool}. \lambda x: \text{Bool}. f (f x)$

$\text{doubleFun} = \lambda f: (\text{Nat} \rightarrow \text{Nat}) \rightarrow (\text{Nat} \rightarrow \text{Nat}).$
 $\lambda x: \text{Nat} \rightarrow \text{Nat}. f (f x)$

- Tienen el **mismo comportamiento**
 - Todas aplican dos veces una función.
- Pero se aplican a **diferentes tipos**.
 - ¡Por lo que son funciones diferentes!
- Ej: Definir una función que aplique dos veces una función
 $(\text{Bool} \rightarrow \text{Bool}) \rightarrow (\text{Bool} \rightarrow \text{Bool})$



Motivación (cont.)

- ▶ Escribir varias veces la misma funcionalidad es mala práctica:
 - ▶ Pérdida de tiempo
 - ▶ Posible introducción de errores
 - ▶ Diferentes nombres para el mismo concepto
- ▶ La única parte que varía son los tipos.
- ▶ La que queremos es **abstraer el tipo**, y luego instanciarlo en cada caso.
- ▶ **Sistema polimórfico**: Cuando podemos usar el mismo fragmento de código con diferentes tipos.

Variedades de Polimorfismo

- ▶ **Polimorfismo Paramétrico:** Un fragmento de código es tipado *genéricamente*, usando variables de tipo en lugar de tipos concretos. Para cada tipo usamos el mismo código. Variantes:
 - ▶ **Polimorfismo de Primera Clase:** Las funciones polimórficas pueden ser pasados como argumentos. No podemos inferir tipos.
 - ▶ **Polimorfismo de `let`:** Restringe el polimorfismo al nivel superior. No se pueden pasar funciones polimórficas como argumento. Admite inferencia de tipos.
- ▶ **Polimorfismo Ad-Hoc:** Distintos comportamiento en diferentes tipos. Ejemplos: sobrecarga de operadores.

Cálculo Lambda Polimórfico

- ▶ Estudiamos el **cálculo λ polimórfico**, también conocido como **Systema F** o cálculo λ de segundo orden.
- ▶ Descubierta por Girard (1972), e independientemente por Reynolds (1974).
- ▶ Polimorfismo paramétrico de primera clase.
- ▶ Extiende el cálculo lambda simple tipado con:
 - ▶ Abstracción de tipos
 - ▶ Aplicación de tipos (instanciación).

Sintaxis del cálculo lambda polimórfico

- ▶ Extendemos los términos

$$t ::= \dots \mid \Lambda X. t \mid t\langle T \rangle$$

- ▶ Extendemos los valores

$$v ::= \dots \mid \Lambda X. t$$

- ▶ Extendemos los tipos

$$T ::= \dots \mid X \mid \forall X. T$$

- ▶ Los contextos Γ ahora pueden ligar variables de tipos.

Reglas Adicionales

- Agregamos reglas de tipado

$$\frac{\Gamma, X \vdash t_2 : T_2}{\Gamma \vdash \Lambda X. t_2 : \forall X. T_2} \quad (\text{T-TABS})$$

$$\frac{\Gamma \vdash t_1 : \forall X. T}{\Gamma \vdash t_1 \langle T_2 \rangle : T [T_2 / X]} \quad (\text{T-TAPP})$$

- Agregamos reglas de evaluación

$$\frac{t_1 \rightarrow t_1'}{t_1 \langle T_2 \rangle \rightarrow t_1' \langle T_2 \rangle} \quad (\text{E-TAPP})$$

$$(\Lambda X. t) \langle T_2 \rangle \rightarrow t [T_2 / X] \quad (\text{E-TAPPABS})$$

Ejemplos

$\text{double}: \forall X. (X \rightarrow X) \rightarrow X \rightarrow X$

$\text{double} = \Lambda X. \lambda f: X \rightarrow X. \lambda x: X. f (f x)$

$\text{doubleNat}: (\text{Nat} \rightarrow \text{Nat}) \rightarrow \text{Nat} \rightarrow \text{Nat}$

$\text{doubleNat} = \text{double} \langle \text{Nat} \rangle$

$\text{doubleBool}: (\text{Bool} \rightarrow \text{Bool}) \rightarrow \text{Bool} \rightarrow \text{Bool}$

$\text{doubleBool} = \text{double} \langle \text{Bool} \rangle$

$\text{doubleFun}: ((\text{Nat} \rightarrow \text{Nat}) \rightarrow \text{Nat} \rightarrow \text{Nat})$

$\rightarrow (\text{Nat} \rightarrow \text{Nat}) \rightarrow \text{Nat} \rightarrow \text{Nat}$

$\text{doubleFun} = \text{double} \langle \text{Nat} \rightarrow \text{Nat} \rangle$

$\text{id}: \forall X. X \rightarrow X$

$\text{id} = \Lambda X. \lambda x: X. x$

Ejercicio: Probar que los términos están bien tipados.

Representando datos

- ▶ Representamos tipos en forma similar a las representaciones de Church del λ -cálculo sin tipos.
 - ▶ Representación de Böhm-Berarducci.
- ▶ Si bien nosotros definimos el cálculo con tipos base (`Nat` y `Bool`), ¡el cálculo lambda polimórfico no los necesita!
- ▶ Las representaciones que veremos son implementables en Haskell con la extensión:

```
{-# LANGUAGE RankNTypes #-}
```

Representando Booleanos

- ▶ La representación de Church de los booleanos:

$\text{verdad} = \lambda t. \lambda f. t$

$\text{falso} = \lambda t. \lambda f. f$

pueden ser tipada con:

$\text{MiBool} = \forall X. X \rightarrow X \rightarrow X$

$\text{verdad} : \text{MiBool}$

$\text{falso} : \text{MiBool}$

- ▶ $\text{not} : \text{MiBool} \rightarrow \text{MiBool}$
 $\text{not} = \lambda b : \text{MiBool}. \Lambda X. \lambda t : X. \lambda f : X. b \langle X \rangle f t$
- ▶ Ejercicio: Definir $\text{and} : \text{MiBool} \rightarrow \text{MiBool} \rightarrow \text{MiBool}$ tal que implemente la conjunción de booleanos.

Representando Naturales

- ▶ Los numerales de Church

$$c_0 = \lambda s. \lambda z. z$$

$$c_1 = \lambda s. \lambda z. s\ z$$

$$c_2 = \lambda s. \lambda z. s\ (s\ z)$$

pueden ser tipados con

$$\text{MiNat} = \forall X. (X \rightarrow X) \rightarrow X \rightarrow X$$

$$c_0, c_1, c_2 : \text{MiNat}$$

$$c_0 = \Lambda X. \lambda s : X \rightarrow X. \lambda z : X. z$$

$$c_1 = \Lambda X. \lambda s : X \rightarrow X. \lambda z : X. s\ z$$

$$c_2 = \Lambda X. \lambda s : X \rightarrow X. \lambda z : X. s\ (s\ z)$$

- ▶ El sucesor es:

$$\text{csucc} : \text{MiNat} \rightarrow \text{MiNat}$$

$$\text{csucc} = \lambda n : \text{MiNat}. \Lambda X. \lambda s : X \rightarrow X. \lambda z : X. s\ (n\langle X \rangle s\ z)$$

Representando Listas

- Representamos listas con el tipo

$$\text{List } X = \forall R. (X \rightarrow R \rightarrow R) \rightarrow R \rightarrow R$$

- Los constructores son:

$$\text{nil} : \forall X. \text{List } X$$

$$\text{nil} = \Lambda X. (\Lambda R. \lambda c : X \rightarrow R \rightarrow R. \lambda n : R. n)$$

$$\text{cons} : \forall X. X \rightarrow \text{List } X \rightarrow \text{List } X$$

$$\text{cons} = \Lambda X. \lambda hd : X. \lambda tl : \text{List } X.$$

$$(\Lambda R. \lambda c : X \rightarrow R \rightarrow R. \lambda n : R. c \text{ } hd \text{ } (tl \langle R \rangle c \text{ } n))$$

$$\text{null} : \forall X. \text{List } X \rightarrow \text{Bool}$$

$$\text{null} = \Lambda X. \lambda xs : \text{List } X. xs \langle \text{Bool} \rangle$$

$$(\lambda hd : X. \lambda tl : \text{List } X. \text{false})$$

$$\text{true}$$

Propiedades Básicas

- ▶ **Preservación:** Si $\Gamma \vdash t : T$ y $t \rightarrow t'$, entonces $\Gamma \vdash t' : T$.
- ▶ **Progreso:** Si t es un término cerrado bien tipado entonces, o bien t es un valor, o bien existe t' tal que $t \rightarrow t'$.
- ▶ **Normalización:** Todo término bien tipado termina.
- ▶ **Inferencia no decidible:** En general, no es posible inferir los tipos de todos los términos. Sin embargo, para determinados términos la inferencia es posible.

- ▶ El polimorfismo permite que un mismo programa se aplique a diferentes tipos.
- ▶ El polimorfismo paramétrico permite que la implementación para los diferentes tipos sea la misma.
- ▶ El lambda cálculo polimórfico tiene polimorfismo paramétrico.
- ▶ Nos permite representar tipos algebraicos de datos.

Referencias

- ▶ *Types and Programming Languages*. B.C. Pierce (2002). Capítulo 23.
- ▶ *Theories of Programming Languages*. J. Reynolds (1998). Capítulo 17.