Polimorfismo

Mauro Jaskelioff

4/10/19

Repaso

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

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

- Los valores son:

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

- Los tipos son

$$T ::= \texttt{Bool} \mid \texttt{Nat} \mid T \to T$$

Reglas de Tipado

$$\Gamma \vdash \texttt{true} : \texttt{Bool} \qquad (\textsc{T-True})$$

$$\Gamma \vdash \texttt{false} : \texttt{Bool} \qquad (\textsc{T-False})$$

$$\frac{\Gamma \vdash t_1 : \texttt{Bool} \quad \Gamma \vdash t_2 : T \quad \Gamma \vdash t_3 : T}{\Gamma \vdash \texttt{if}\ t_1\ \texttt{then}\ t_2\ \texttt{else}\ t_3 : T} \qquad (\textsc{T-If})$$

$$\Gamma \vdash 0 : \texttt{Nat} \qquad (\textsc{T-Zero})$$

$$\frac{\Gamma \vdash t_1 : \texttt{Nat}}{\Gamma \vdash \texttt{succ}\ t_1 : \texttt{Nat}} \qquad (\textsc{T-Succ})$$

$$\frac{\Gamma \vdash t_1 : \texttt{Nat}}{\Gamma \vdash \texttt{pred}\ t_1 : \texttt{Nat}} \qquad (\textsc{T-Pred})$$

Reglas de Tipado (cont.)

$\vdots$

$$\frac{\Gamma \vdash t_1 : \mathtt{Nat}}{\Gamma \vdash \mathtt{iszero}\ t_1 : \mathtt{Bool}} \qquad (\text{T-IsZero})$$

$$\frac{x : T \in \Gamma}{\Gamma \vdash x : T} \qquad (\text{T-Var})$$

$$\frac{\Gamma, x : T_1 \vdash t_2 : T_2}{\Gamma \vdash \lambda x : T_1.\ t_2 : T_1 \to T_2} \qquad (\text{T-Abs})$$

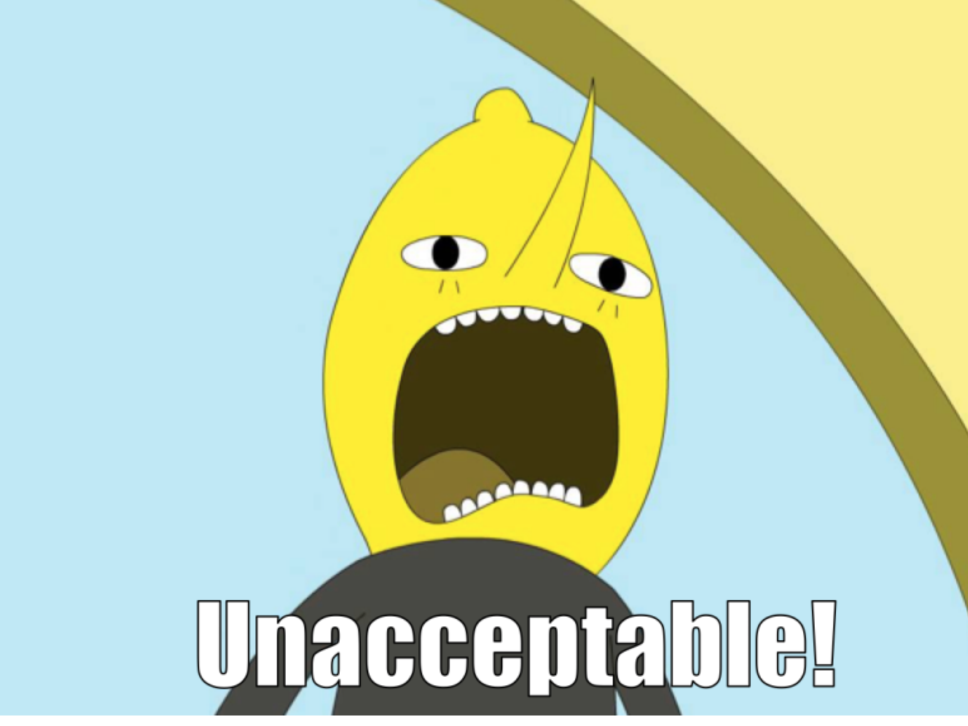
$$\frac{\Gamma \vdash t_2 : T_1 \to T_2 \quad \Gamma \vdash t_1 : T_1}{\Gamma \vdash t_2\ t_1 : T_2} \qquad (\text{T-App})$$

Motivación

- Consideremos las funciones

$$\begin{aligned}
\texttt{doubleNat} &= \lambda f : \texttt{Nat} \to \texttt{Nat}.\, \lambda x : \texttt{Nat}.\ f\ (f\ x) \\
\texttt{doubleBool} &= \lambda f : \texttt{Bool} \to \texttt{Bool}.\, \lambda x : \texttt{Bool}.\ f\ (f\ x) \\
\texttt{doubleFun} &= \lambda f : (\texttt{Nat} \to \texttt{Nat}) \to (\texttt{Nat} \to \texttt{Nat}). \\
&\qquad\qquad\qquad\qquad \lambda x : \texttt{Nat} \to \texttt{Nat}.\ f\ (f\ x)
\end{aligned}$$

- 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 $(\texttt{Bool} \to \texttt{Bool}) \to (\texttt{Bool} \to \texttt{Bool})$


Unacceptable!

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`**: Restringue 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 **Sistema F** o cálculo λ de segundo orden.
- Descubierto 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

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

- Extendemos los valores

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

- Extendemos los tipos

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

- Los contextos Γ ahora pueden ligar variables de tipos.

Reglas Adicionales

- Agregamos reglas de tipado

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

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

- Agregamos reglas de evaluación

$$\frac{t_1 \to t_1'}{t_1\langle T_2 \rangle \to t_1'\langle T_2 \rangle} \qquad (\text{E-TApp})$$

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

Ejemplos

$$\texttt{double} \colon \forall X.\ (X \to X) \to X \to X$$
$$\texttt{double} = \Lambda X. \lambda f \colon X \to X. \lambda x \colon X.\ f\ (f\ x)$$

$$\texttt{doubleNat} \colon (\texttt{Nat} \to \texttt{Nat}) \to \texttt{Nat} \to \texttt{Nat}$$
$$\texttt{doubleNat} = \texttt{double}\langle\texttt{Nat}\rangle$$

$$\texttt{doubleBool} \colon (\texttt{Bool} \to \texttt{Bool}) \to \texttt{Bool} \to \texttt{Bool}$$
$$\texttt{doubleBool} = \texttt{double}\langle\texttt{Bool}\rangle$$

$$\begin{aligned}\texttt{doubleFun} \colon & ((\texttt{Nat} \to \texttt{Nat}) \to \texttt{Nat} \to \texttt{Nat}) \\ & \to (\texttt{Nat} \to \texttt{Nat}) \to \texttt{Nat} \to \texttt{Nat}\end{aligned}$$
$$\texttt{doubleFun} = \texttt{double}\langle\texttt{Nat} \to \texttt{Nat}\rangle$$

$$\texttt{id} \colon \forall X.\ X \to X$$
$$\texttt{id} = \Lambda X. \lambda x \colon X.\ x$$

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 #-}
  ```

  pero el operador Λ y la instanciación de tipos $\langle\rangle$ son invisibles.

Representando Booleanos

- La representación de Church de los booleanos:

  $$\texttt{verdad} = \lambda t.\, \lambda f.\; t \qquad \texttt{falso} = \lambda t.\, \lambda f.\; f$$

  puede ser tipada con:

  $$\begin{aligned}
  &\texttt{MiBool} = \forall X.\; X \to X \to X \\
  &\texttt{verdad}, \texttt{falso} : \texttt{MiBool} \\
  &\texttt{verdad} = \Lambda X.\, \lambda t : X.\, \lambda f : X.\; t \\
  &\texttt{falso} \;\; = \Lambda X.\, \lambda t : X.\, \lambda f : X.\; f
  \end{aligned}$$

- $$\begin{aligned}
  &\texttt{not} : \texttt{MiBool} \to \texttt{MiBool} \\
  &\texttt{not} = \lambda b : \texttt{MiBool}.\; \Lambda X.\, \lambda t : X.\, \lambda f : X.\; b\langle X \rangle f\; t
  \end{aligned}$$

- Ejercicio: Definir *and* : $\texttt{MiBool} \to \texttt{MiBool} \to \texttt{MiBool}$ tal que implemente la conjunción de booleanos.

Representando Naturales

- Los numerales de Church

$$\begin{aligned} c_0 &= \lambda s.\, \lambda z.\;\; z \\ c_1 &= \lambda s.\, \lambda z.\;\; s\ z \\ c_2 &= \lambda s.\, \lambda z.\;\; s\ (s\ z) \end{aligned}$$

  puden ser tipados con

$$\texttt{MiNat} = \forall X.\ (X \to X) \to X \to X$$

$$\begin{aligned} & c_0, c_1, c_2 \colon \texttt{MiNat} \\ c_0 &= \Lambda X.\, \lambda s \colon X \to X.\, \lambda z \colon X.\;\; z \\ c_1 &= \Lambda X.\, \lambda s \colon X \to X.\, \lambda z \colon X.\;\; s\ z \\ c_2 &= \Lambda X.\, \lambda s \colon X \to X.\, \lambda z \colon X.\;\; s\ (s\ z) \end{aligned}$$

- El sucesor es:

$$\begin{aligned} & \texttt{csucc} \colon \texttt{MiNat} \to \texttt{MiNat} \\ & \texttt{csucc} = \lambda n \colon \texttt{MiNat}.\ \Lambda X.\ \lambda\, s \colon X \to X.\, \lambda z \colon X.\;\; s\ (n\langle X \rangle s\ z) \end{aligned}$$

Representando Listas

- Representamos listas con el tipo

$$\texttt{List}\ X = \forall R.\ (X \to R \to R) \to R \to R$$

- Los constructores son:

$$\begin{aligned}
&\mathit{nil} : \forall X.\ \texttt{List}\ X \\
&\mathit{nil} = \Lambda X.\ (\Lambda R.\, \lambda c : X \to R \to R.\, \lambda n : R.\ n)
\end{aligned}$$

$$\begin{aligned}
&\mathit{cons} : \forall X.\ X \to \texttt{List}\ X \to \texttt{List}\ X \\
&\mathit{cons} = \Lambda X.\, \lambda hd : X.\, \lambda tl : \texttt{List}\ X. \\
&\qquad\qquad (\Lambda R.\, \lambda c : X \to R \to R.\, \lambda n : R.\ c\ hd\ (tl\langle R\rangle c\ n))
\end{aligned}$$

$$\begin{aligned}
&\mathit{null} : \forall X.\ \texttt{List}\ X \to \texttt{Bool} \\
&\mathit{null} = \Lambda X.\, \lambda xs : \texttt{List}\ X.\ xs\langle\texttt{Bool}\rangle \\
&\qquad\qquad\qquad\qquad (\lambda hd : X.\, \lambda tl : \texttt{Bool}.\ \texttt{false}) \\
&\qquad\qquad\qquad\qquad \texttt{true}
\end{aligned}$$

Propiedades Básicas

- **Preservación**: Si $\Gamma \vdash t : T$ y $t \to t'$, 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 $t \to t'$.
- **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.

Resumen

- 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.