Programación Funcional

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

Definición de funciones

Definición de funciones

Notación similar a la usada en matemática.

$$square :: Integer \to Integer$$
$$square\ x = x * x$$

¿Cómo hago para definir funciones por casos?

$$abs(x) = \begin{cases} x, & \text{si } x \geq 0 \\ -x, & \text{en otro caso.} \end{cases}$$

Ecuaciones condicionales

- Es posible definir funciones a través de **ecuaciones con condiciones**.

$$\begin{array}{lll} abs\ x & \mid x \geqslant 0 & = x \\ & \mid otherwise & = -x \end{array}$$

  donde *otherwise* es una condición que siempre es verdadera.

- Las condiciones se evaluan en secuencia y se toma la primera opción cuya condición sea verdadera.

- Si todas las condiciones fallan, entonces da error.

- Una forma alternativa de definición es mediante el uso de **if**.

$$abs\ x = \text{if } x \geqslant 0 \text{ then } x \text{ else } -x$$

Pattern matching

- Las funciones pueden ser definidas por casos a través del uso de **patrones**.

$$
\begin{aligned}
&\mathit{not}\ \mathit{True} = \mathit{False}\\
&\mathit{not}\ \mathit{False} = \mathit{True}\\
&\\
&\mathit{True}\ \&\&\ b = b\\
&\mathit{False}\ \&\&\ \_ = \mathit{False}
\end{aligned}
$$

- Los patrones satisfacen la siguiente gramática:

$$
\begin{aligned}
\mathit{pat} ::= {} & \_ && \text{-- wildcard}\\
\mid {} & \mathit{variable}\\
\mid {} & \mathit{literal}\\
\mid {} & (\mathit{pat}_1, ..., \mathit{pat}_m)\\
\mid {} & \mathit{pat} : \mathit{pat}\\
\mid {} & C\ \mathit{pat}_1\ ...\ \mathit{pat}_n && \text{-- C denota un constructor}\\
\mid {} & \mathit{var}@\mathit{pat} && \text{-- as pattern}
\end{aligned}
$$

Pattern matching: ejemplos

$$
\begin{aligned}
&\mathit{fst}\ (x, y) &&= x \\
&\mathit{fst}\ (x, \_) &&= x \\
&\mathit{snd}\ (\_, y) &&= y \\
&\mathit{head}\ (x\colon \_) &&= x \\
&\mathit{tail}\ \ (\_ : xs) &&= xs
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{duplica}\ (x : xs) &&= x : x : xs \\
&\mathit{duplica}\ s@(x : xs) &&= x : s
\end{aligned}
$$

$$
\begin{aligned}
&\text{data}\ \mathit{AoB} = \mathit{A}\ \mathit{Int} \mid \mathit{B} \\
&g :: \mathit{AoB} \to \mathit{Int} \\
&g\ (\mathit{A}\ 4) = 8 \\
&g\ (\mathit{A}\ x) = x \\
&g\ \mathit{B} \quad\ = 0
\end{aligned}
$$

Preguntas

Definiciones locales (where)

- **Definiciones locales** pueden ser introducidas por el uso de la palabra reservada **where**.

$$
\begin{aligned}
&f\ x\ y = (a+1) * (a+2) \\
&\quad \text{where} \\
&\qquad a = (x+y)\ /\ 2
\end{aligned}
$$

El alcance de dicha definición local es la expresión en la parte derecha de la definición de f.

- Pueden haber múltiples definiciones locales:

$$
\begin{aligned}
&f\ x\ y = (a+1) * (b+2) \\
&\quad \text{where} \\
&\qquad a = (x+y)\ /\ 2 \\
&\qquad b = (x+y)\ /\ 3
\end{aligned}
$$

Definiciones locales (where)

Definiciones locales pueden ser usadas junto con ecuaciones condicionales.

$$
\begin{array}{l}
f\ x\ y \mid p < 1 \quad\quad = x + p \\
\qquad\ \mid p > 1 \quad\quad = x - p \\
\qquad\ \mid otherwise = x \\
\quad \text{where} \\
\qquad p = x\ /\ (y + 1)
\end{array}
$$

La definición local es visible en toda la ecuación, incluso en sus condiciones.

Definiciones locales (let)

- Una forma alternativa de escribir definiciones locales es mediante el uso de **let**:

$$f\ x\ y = \text{let } a = (x + y)\ /\ 2 \\ \text{in } (a + 1) * (a + 2)$$

  El alcance de las definiciones locales es la expresión en el **in**.

- A diferencia del **where** un let . . . in . . . es una expresión.

Ejemplos let

let $x = 2$ in let $y = x + 3$ in $y * 4$

$\Rightarrow$ evalúa a 20

let $x = 2$ in $x * 3 +$ let $y = 3$ in $x + y$

$\Rightarrow$ evalúa a 11

(let $x = 2$ in $x * 3$) + let $y = 3$ in $x + y$

$\Rightarrow$ error!: x no es visible en el segundo **let**

Preguntas

Operadores

- A las funciones definidas en forma infija se le suele llamar **operadores**.

$$3 + 4$$
$$7 * 8$$
$$2 \leqslant 3$$

- Al escribir un operador entre paréntesis lo convertimos en una función prefija currificada:

$$(+)\ 3\ 4$$
$$(*)\ \ 7\ 8$$
$$(\leqslant)\ 2\ 3$$

De esta forma el operador puede ser pasado como argumento a funciones:

$$\mathit{suma} = \mathit{uncurry}\ (+)$$
$$\mathit{leq} \quad = \mathit{uncurry}\ (\leqslant)$$

Operadores (2)

- La aplicación de un operador en forma infija requiere que el mismo esté saturado, esto es, que se le pasen todos sus argumentos.

  La expresión $3 + 4$ es correcta, mientras que $3+$ no lo es.

- En cambio, al escribir un operador entre paréntesis es posible parametrizarlo parcialmente:

  $(+)\ 3$
  $(*)\ 7$
  $(\leqslant)\ 2$

Secciones

- Las **secciones** son una sintaxis especial para expresar la parametrización parcial de un operador infijo.
- Se encierra entre paréntesis el operador junto a uno de sus argumentos.

$$(3+)$$
$$(7*)$$
$$(2 \leqslant)$$
$$(*4)$$
$$(>0)$$

- Ejemplos de uso:

$$\mathit{filter}\ (>0)$$
$$\mathit{map}\ (+1)$$
$$(3+)\ .\ \mathit{abs}$$

Preguntas

Definiciones anónimas

- Es posible definir **funciones anónimas** (o sea, sin nombre) mediante el uso de las llamadas expresiones lambda.

  $$\lambda x \to x + 1$$

  En Haskell se escribe: `\x -> x + 1`.

- La aplicación de una función anónima a sus argumentos se realiza de la misma forma que con las otras funciones:

  $$(\lambda x \to x + 1)\ 2$$

- Las definiciones de funciones con nombre también pueden ser vistas como definiciones en términos de expresiones lambda:

  $$\begin{aligned} \mathit{suma1} &= \lambda x \to x + 1 \\ \mathit{add} &= \lambda x \to (\lambda y \to x + y) \end{aligned}$$

Preguntas