Programación Funcional

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

Tipos Algebraicos

Tipos en Haskell

- Tipos Básicos (*Int*, *Integer*, *Float*, *Bool*, *Char*)
- Tipos Compuestos
  - Tuplas : $(t_1, .., t_n)$
  - Listas : $[t_1]$
  - Función : $t_1 \rightarrow t_2$
- Tipos Algebraicos

Tipos Algebraicos

- Una primera definición...

$$\begin{aligned}\text{data } T = & \; C_1 \; t_{11} \; ... \; t_{1k_1} \\ & \; ... \\ | & \; C_n \; t_{n1} \; ... \; t_{nk_n}\end{aligned}$$

  donde T es el nuevo tipo a introducir y cada C_i es un constructor de dicho tipo, que toma k_i parámetros.

- Un elemento de tipo T se construye aplicando alguno de sus constructores C_i a argumentos con tipos acordes a los dados en la definición

$$C_i \; v_{i1} \; ... \; v_{ik_i}$$

- Los constructores se pueden ver como funciones de tipo:

$$C_i :: t_{i1} \rightarrow ... \rightarrow t_{ik_i} \rightarrow T$$

Tipos Algebraicos Recursivos

- El tipo que se está definiendo se puede usar como tipo componente en la definición.

- Ejemplos:
  - Listas de caracteres

$$\text{data } \mathit{LChar} = \mathit{Nil} \mid \mathit{Cons}\ \mathit{Char}\ \mathit{LChar}$$

  - Árboles Binarios de Enteros

$$\text{data } \mathit{Tree} = \mathit{Empty} \mid \mathit{Branch}\ \mathit{Int}\ \mathit{Tree}\ \mathit{Tree}$$

  - Expresiones Numéricas

$$\begin{aligned}\text{data } \mathit{Expr} = {} & \mathit{Lit}\ \mathit{Int} \\ \mid {} & \mathit{Add}\ \mathit{Expr}\ \mathit{Expr} \\ \mid {} & \mathit{Sub}\ \mathit{Expr}\ \mathit{Expr}\end{aligned}$$

Constructores Infijos

- Al igual que con las funciones, los constructores se pueden escribir de forma infija:

$$expr = (Lit\ 2)\ `Add`\ (Lit\ 4)$$

- Pero también se pueden definir **constructores infijos**:

$$\begin{aligned} \text{data } Expr = &\ Lit\ Int \\ \mid &\ Expr\ :+:\ Expr \\ \mid &\ Expr\ :-:\ Expr \end{aligned}$$

Los constructores tienen que empezar con ':'.

Preguntas

Recursión Estructural

- Muchas funciones pueden implementarse utilizando **recursión estructural**

- Siguiendo el esquema:
  - En los casos base (no recursivos) se puede utilizar el valor
  - En los casos recursivos se pueden utilizar los resultados de las llamadas recursivas de las ocurrencias recursivas del tipo

- Ejemplos:
  - Evaluar la expresión

$$
\begin{aligned}
&eval :: Expr \to Int \\
&eval\ (Lit\ \ n) &&= n \\
&eval\ (Add\ e1\ e2) &&= eval\ e1 + eval\ e2 \\
&eval\ (Sub\ e1\ e2) &&= eval\ e1 - eval\ e2
\end{aligned}
$$

  - Convertir expresiones a strings, contar operadores de una expresión, altura de un árbol, sumar elementos del árbol, etc.

Recursión General

- Hay funciones que no se pueden definir utilizando recursión estructural

- Ejemplo: parentizar las sumas de una expresión hacia la derecha.
  - $(2+3)+4 \rightsquigarrow 2+(3+4)$
  - $((2+3)+4)+5 \rightsquigarrow 2+(3+(4+5))$
  - $((2-((6+7)+8))+4)+5 \rightsquigarrow (2-(6+(7+8)))+(4+5)$

- Ejemplos:

$$
\begin{aligned}
&\mathit{assoc} :: \mathit{Expr} \to \mathit{Expr} \\
&\mathit{assoc}\ (\mathit{Add}\ (\mathit{Add}\ e1\ e2)\ e3) = \mathit{assoc}\ (\mathit{Add}\ e1\ (\mathit{Add}\ e2\ e3)) \\
&\mathit{assoc}\ (\mathit{Add}\ e1\ e2) \quad\quad\quad\quad = \mathit{Add}\ (\mathit{assoc}\ e1)\ (\mathit{assoc}\ e2) \\
&\mathit{assoc}\ (\mathit{Sub}\ e1\ e2) \quad\quad\quad\quad = \mathit{Sub}\ (\mathit{assoc}\ e1)\ (\mathit{assoc}\ e2) \\
&\mathit{assoc}\ (\mathit{Lit}\ n) \quad\quad\quad\quad\quad\quad = \mathit{Lit}\ n
\end{aligned}
$$

Recursión Mutua

- Al definir un tipo se puede utilizar otros tipos que a su vez refieren al tipo original.

$$\begin{aligned}\text{data } &Tree \;\; = Empty \mid Node\ Int\ Forest\\ \text{data } &Forest = Nil \mid Cons\ Tree\ Forest\end{aligned}$$

- Funciones mutuamente recursivas

$$\begin{aligned}&flattenTree :: Tree \to [Int]\\ &flattenTree\ Empty \;\;\;\;\;\;\;\;\;\;\;\; = [\,]\\ &flattenTree\ (Node\ x\ xs) = x : flattenForest\ xs\\ &flattenForest :: Forest \to [Int]\\ &flattenForest\ Nil \;\;\;\;\;\;\;\;\;\;\;\;\;\;\; = [\,]\\ &flattenForest\ (Cons\ x\ xs) = flattenTree\ x +\!\!+ flattenForest\ xs\end{aligned}$$

Preguntas