El Sistema de Tipos de Haskell: Tipos Básicos

Tecnología de Programación

Adolfo Muñoz – Juan Magallón
Grado en Ingeniería Informática

Universidad Zaragoza
1542

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza
1542

2023-01-20

Objetivos

Objetivos

El objetivo de este tema es presentar:

- la organización de los tipos básicos de Haskell
- la forma de ampliar ese conjunto de tipos

Introducción

El **Sistema de Tipos** de un lenguaje de programación es el conjunto de reglas que permite comprender el funcionamiento de los tipos de datos en dicho lenguaje y la forma de ampliar ese conjunto de tipos, es decir, definir nuevos tipos de datos.

- El nombre de un tipo de datos siempre comienza por una letra mayúscula:

  `Bool, Integer, DiaDeLaSemana`

- En Haskell los tipos se utilizan para:
  - Definir parámetros y resultado de funciones
  - Acotar la interpretación de una expresión, mediante el operador `'::'`, en el caso de que el resultado sea polimórfico y no quede determinado por el contexto.

  Ambas cosas pueden considerarse el equivalente a una *declaración* en otros lenguajes.

  Pero **NO** son declaraciones.

- Definir parámetros y resultado de funciones...

```
1  sin :: Float -> Float
2  add :: Int -> Int -> Int
```

... y operadores

```
1  (+) :: Float -> Float -> Float
2  (^) :: Float -> Int -> Float
```

- Acotar la interpretación de una expresión:

```
ghci> a = 1
ghci> a :: Int
1
ghci> a :: Double
1.0

ghci> False && (read "True")
False
ghci> read "True"
*** Exception: Prelude.read: no parse
ghci> read "True" :: Bool
True
```

En la definición de una función pueden existir tipos polimórficos, que se denominan **tipos libres** o **variables de tipo**:

```
1  add  :: a -> a -> a
2  read :: String -> a
3  show :: a -> String
4  head :: [a] -> a
5  map  :: (a -> b) -> [a] -> [b]
```

Tipos simples

Valores:

- `True`
- `False`

Operadores:

- `(&&)  :: Bool -> Bool -> Bool`
- `(||)  :: Bool -> Bool -> Bool`
- `not  :: Bool -> Bool`
- `otherwise :: Bool`
  `otherwise = True`

Valores:

- **Int**: Tipo entero de precisión limitada (64 bits).
- **Integer**: Tipo entero con precisión infinita.

Operaciones:

- `(+) (-) (*) (^) :: Int -> Int -> Int`
- `div mod :: Int -> Int -> Int`
- `abs negate (-) :: Int -> Int`
- `even odd :: Int -> Bool`
- `fromInteger :: Int -> a`

Valores:

- Tipo flotante de precisión limitada.

Operaciones:

- `(+) (-) (*) (/) (**) :: Float -> Float -> Float`
- `(^)                  :: Float -> Int -> Float`
- `abs negate (-)       :: Float -> Float`
- `sin asin cos acos tan atan atan2 log exp sqrt`
- `truncate round floor ceiling`
- `pi :: Float`

Valores:

- Caracteres.

Operaciones: (`import Data.Char`)

- `ord :: Char -> Int`
- `chr :: Int -> Char`
- `isUpper isLower :: Char -> Bool`
- `isDigit isAlpha :: Char -> Bool`
- `toUpper toLower :: Char -> Char`

Tipos enumerados

La definición de los tipos enumerados es sencilla:

```
1 data Bool   = False | True
2 data Dia    = Lun | Mar | Mie | Jue | Vie | Sab | Dom
3 data State  = ON | OFF
```

...pero más tarde vermos que hay mucha más complejidad escondida en esa definición.

Una versión más completa sería algo tipo:

```
1 data State  = ON | OFF deriving (Enum,Eq,Ord)
```

También puede definirse un nuevo tipo de datos con múltiples opciones:

```haskell
data LetraONumero = Letra Char | Numero Integer

siguiente :: LetraONumero -> LetraONumero
siguiente (Numero n) = Numero (1 + n)
siguiente (Letra c)  = Letra (chr (1 + ord c))
```

Profundizaremos más adelante.

Para todos los tipos básicos están definidos los operadores de igualdad y comparación

- `== /=`
- `< >`
- `<= >=`
- `min max`

Tipos estructurados

En otros lenguajes se conocen como registros anónimos.

Ejemplos:

- `()` `:: ()` (tupla vacía)
- `('a',True)` `:: (Char,Bool)`
- `('a',True,1.5)` `:: (Char,Bool,Float)`

Operadores:

- `fst :: (a,b) -> a`
- `snd :: (a,b) -> b`

Las tuplas son útiles cuando una función debe devolver más de un valor:

```
1  sincos :: Float -> (Float,Float)
2  sincos x = (sin x,cos x)
3
4  sc = sincos pi
5  s  = fst sc
6  c  = snd sc
7
8  main = do
9           print s
10          print c
```

Avance: desempaquetado de tuplas mediante ajuste de patrones:

```
1  sincos :: Float -> (Float,Float)
2  sincos x = (sin x,cos x)
3
4  (s,c) = sincos pi
5
6
7
8  main = do
9           print s
10          print c
```

Las listas son colecciones homogéneas de elementos.
Hay dos constructores básicos para listas (*ala* TAD...):

- `[]` construye una lista vacía
- `(:)` construye una lista a partir de un elemento y otra lista

A partir de ellos se puede construir cualquier otra lista:

```
1:[]
1:(2:(3:[]))
```

...pero la notación es farragosa y existen notaciones simplificadas:

```
1:2:3:[]  -- el operador : es asociativo por la derecha
[1,2,3]   -- notacion simplificada
```

Los cadenas de caracteres son simplemente listas de caracteres:

```
1 String = [Char]
2 ['h','o','l','a'] = "hola"
```

El tipo String es simplemente un alias a [Char].

Referencias:

- **Hoogle**
  `https://hoogle.haskell.org/`
- **Prelude**
  `http://www.haskell.org/onlinereport/prelude-index.html`

El Sistema de Tipos de Haskell: Tipos Básicos

Tecnología de Programación

Adolfo Muñoz – Juan Magallón
Grado en Ingeniería Informática

Universidad Zaragoza
1542

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Departamento de Informática e Ingeniería de Sistemas
Universidad Zaragoza
1542

2023-01-20