Programación con tipos dependientes en Idris 2

Sintaxis básica I

Axel Suárez Polo

March 12, 2021

BUAP

Contenidos

Introducción

¿Qué es Idris?

- Es un lenguaje funcional **puro** con **tipos dependientes**.
- Tiene un sistema de tipos similar al de **Agda**.
- Tiene una sintaxis y manejo de efectos similar a **Haskell**
- Busca ser un lenguaje práctico pero con tipos dependientes.

Un ejemplo de Idris

```
module Main

main : IO ()
main = putStrLn "Hello World"
```

Listing 1: Hello world en Idris

Sintaxis básica

Comentarios

En Idris existen 3 tipos de comentarios:

- Comentarios de una sóla línea:

```
1  -- este es un comentario
2  -- los comentarios comienzan con "--"
```

- Comentarios multilínea:

```
1  {- este es un comentario
2  que se puede expandir por múltiples líneas
3  -}
```

- Comentarios de documentación:

```
1  ||| Este comentario documenta una declaración
2  ||| Como ejemplo, main
3  main : IO ()
4  main = putStrLn "Hello World"
```

Top-level declarations

- En Idris sólo existen declaraciones *top-level* y *locales*.
- Las declaraciones *top-level* tienen la siguiente sintaxis:

```
1  -- <identificador> : <tipo>
2  -- <identificador> = <expresión>
3  x : Int
4  x = 10
```

Listing 2: Sintaxis de las declaraciones *top-level*.

- Los **identificadores** deben coincidir.
- La **expresión** debe ser del **tipo** especificado.

[1] **Diferencias con Haskell**: La anotación de tipo es obligatoria y se utiliza `:` en lugar de `::`

Identificadores

- Los identificadores válidos son secuencias de caracteres que:
  - **Comienzan con una letra** ya sea minúscula o mayúscula.
  - Seguido de 0 o más letras, números, apóstrofes o guiones bajos.
  - Terminan en espacio en blanco o algún carácter que no esté en la lista anterior.
- Identificadores válidos: *x, y, var_, Test, x"3,* $\alpha\xi\epsilon\lambda$
- Identificadores inválidos: *3x, x-y, _ok, ?ok*

[1] **Diferencias con Haskell**: Las variables también pueden iniciar con mayúsula sin ser tipos.

Tipos

Los tipos son más complejos y existe una gran diversidad de estos:

- Tipos
  - **Primitivos**
    - **Numéricos**
    - **De texto**
  - Definidos por el usuario
    - Records
    - ADTs (Algebraic Data Types)
    - GADTs
  - Compuestos
    - **Tuplas**
    - **Funciones***
    - Higher Kinded Types
  - Dependientes
    - **Π**
    - **Σ**

Expresiones

Las expresiones, al igual que los tipos, son bastante diversas:

- Expresiones
  - Literales
    - Enteras
    - Punto flotante
    - Texto
    - Lista
    - Función (lambdas)*
  - Aplicación de funciones*
    - Operadores y secciones*
  - Identificadores
  - Azúcar sintáctica
    - `let`, `where`, `case`, `if`, `do`, `rewrite`
    - Comprensiones de mónadas y !-notation
    - Idiom brackets para aplicativos
  - Holes

Tipos numéricos

En Idris existen los tipos númericos a los que estamos acostumbrados de otros lenguajes y algunos nuevos:

- `Int`: Entero de precisión fija, con al menos 32 bits.
- `Integer`: Entero de precisión no acotada.
- `Nat`: Entero sin signo de precisión no acotada.
- `Double`: Número de punto flotante de doble precisión.

Literales numéricas

Para obtener un valor de alguno de los tipos mencionados, podemos utilizar literales:

- Literales de enteros no-negativos para **`Int`**, **`Integer`** y **`Nat`**:
  - `202103 -- decimal`
  - `0b0101 -- binario`
  - `0o2321 -- octal`
  - `0xefff -- hexadecimal`
- Literales de enteros para **`Int`** y **`Integer`**:
  - *Todas las anteriores, y además con un '-' al inicio*
  - `-202103`
  - `-0xefff`

Literales numéricas (continuación)

- Literales de números de punto flotante para **`Double`**:
  - *Todas las anteriores*
  - `2021.0312 -- decimal con punto`
  - `0.2021e-3 -- científica`

Ejemplos literales numéricas

```
x : Int
x = 10

y : Double
y = 12.2

z : Nat
z = 0b10101
```

Listing 3: Literales numéricas en Idris

```
int x = 10;

double y = 12.2;

// Más cercano
var z = BigInteger.valueOf(0b10101);
```

Listing 4: Literales numéricas en Java

Tipos de texto

También existen los tipos para manejar texto a los que estamos acostumbrados:

- **`Char`**: Carácter Unicode (code point)
- **`String`**: Secuencia de longitud fija de **`Char`**'s

Literales de texto

Al igual que los tipos numéricos, los tipos de texto tienen literales:

- Literales de carácter para **`Char`**:
  - `'a'    -- carácter`
  - `'λ'    -- carácter no ascii`
  - `'\97'  -- literal decimal`
  - `'\x61' -- literal hexadecimal`
- Literales de cadena **`String`**:
  - `"letras" -- secuencias de caracteres`
  - `"letr\97s"`
  - ```
    " multiline
    string
    " -- pueden ocupar múltiples líneas
    ```
    ```

Tipos compuestos: Tuplas

- El tipo compuesto más simple es la **tupla**.
- Una **tupla** es una colección ordenada de tamaño fijo que puede ser heterogénea.
- La sintaxis para las tuplas en Idris es una lista de 2 o más valores separados por comas y rodeados por paréntesis.

```
persona : (String, Double)
persona = ("Juan", 9.8)

-- Pueden tener más de 2 valores
fecha : (Nat, Nat, Nat)
fecha = (12, 3, 2021)

-- Pueden anidarse y el anidamiento es significativo
exposición : ((String, Double), (Nat, Nat, Nat))
exposición = (persona, fecha)
-- exposición = (("Juan", 9.8), (12, 3, 2021))
```

Listing 5: Tuplas en Idris

Tipos compuestos: Tuplas (continuación)

- Hay propiedades que no tienen las tuplas pero uno podría inferir *incorrectamente* que tienen:
  - No conmutan en general:

    ```
    (String, Double) /= (Double, String)
    ```
  - No asocian:

    ```
    (String, (Double, Nat)) /= ((String, Double), Nat)
                            /= (String, Double, Nat)
    ```
- Sin embargo, los valores de estos tipos contienen la misma cantidad de información, con lo que se pueden construir **isomorfismos** entre ellos.

```
(String, Double) ~ (Double, String)
("Juan", 9.8) ~ (9.8, "Juan")
```

Tipos compuestos: Funciones

- Uno de los tipos compuestos más útiles y centrales a la programación funcional es el de **tipo de función**.
- Un **tipo de función** está compuesto de uno o más *tipos de entrada* y un único *tipo de salida*.
- La sintaxis para representar un tipo de función es mediante una lista ordenada de dos o más tipos separados por `->`
  (un guión medio y un signo de mayor que).

```
cuadrado : Double -> Double

edad : (String, Nat) -> Nat

sumar : Int -> Int -> Int

dividirEn : Int -> String -> (String, String)
```

Listing 6: Tipos de función en Idris

Tipos compuestos: Funciones (continuación)

- Al igual que con las tuplas, los tipos de función tienen algunas propiedades:
  - **No conmutan en general:**

    ```
    String -> Int /= Int -> String
    ```
  - **Asocian sólo a la derecha:** A diferencia de las tuplas, los tipos de función si presentan asociatividad pero sólo hacia la derecha:

    ```
    Int -> Int -> Int == Int -> (Int -> Int)
    Int -> Int -> Int /= (Int -> Int) -> Int
    ```

    Una forma de no perderse es recordar que los tipos de función “marcan” a los tipos como de entrada o de salida, por lo que un tipo de entrada nunca puede convertirse en uno de salida.

Tipos compuestos: Funciones (continuación)

- Al igual que con las tuplas, los tipos de función tienen algunas propiedades:
  - No conmutan en general:

    ```
    String -> Int /= Int -> String
    ```

  - Asocian sólo a la derecha: A diferencia de las tuplas, los tipos de función si presentan asociatividad pero sólo hacia la derecha:

    ```
    --I     I      O      I       I      O
    Int -> Int -> Int == Int -> (Int -> Int)
    --I     I      O      I       O      O
    Int -> Int -> Int /= (Int -> Int) -> Int
    ```

    Una forma de no perderse es recordar que los tipos de función “marcan” a los tipos como de entrada o de salida, por lo que un tipo de entrada nunca puede convertirse en uno de salida.

Tipos compuestos: Funciones (continuación)

- También presentan **isomorfismos** que más adelante serán vistos:

```
-- atestiguado por flip
Int -> Double -> String  ~  Double -> Int -> String
-- atestiguado por curry y uncurry
(Int, Double) -> String  ~  Int -> Double -> String
```

Funciones sobre tipos numéricos

- Idris incluye por defecto múltiples funciones para trabajar sobre tipos numéricos:

```
(+) : Int -> Int -> Int
(*) : Int -> Int -> Int
```

- Estas mismas funciones existen para los demás tipos numéricos: `Nat`, `Integer` y `Double`.
- También se incluyen la función de resta, pero no está definida para `Nat`.

```
(-) : Int -> Int -> Int
```

Funciones sobre tipos numéricos (continuación)

- Además, define la operación de división entera y módulo para los tipos enteros (`Int` e `Integer`):

```
div : Int -> Int -> Int
mod : Int -> Int -> Int
```

- Finalmente, también define la división de números de punto flotante:

```
(/) : Double -> Double -> Double
```

Aplicación de funciones

- Lo más relevante que se puede hacer con una función es aplicarla.
- Como esto es lo más común de hacer, en Idris se proporciona la sintaxis de aplicación de función que consiste en lo siguiente:
  - Empieza con una **expresión** del tipo función.
  - Es seguida por 1 o más expresiones separadas por espacios en blanco, que serán los argumentos para la función.
  - Una **expresión de aplicación función** es válida cuando:
    - se proporcionan expresiones con los tipos correspondientes a los parámetros que declara la función.
    - se proporciona una cantidad de argumentos **menor o igual** a la cantidad de parámetros que declara la función.

Ejemplos de aplicación funciones

```
sumar : Int -> Int -> Int
sumar = (+)

x : Int
x = 10

y : Int
y = 20

z : Int
z = sumar x y -- 30

z' : Int
z' = sumar 10 20
```

Listing 7: Aplicación de funciones en Idris

```
int sumar(int a, int b) {
  return a + b;
}

int x = 10;

int y = 20;


int z = sumar(x, y);


int zp = sumar(10, 20);
```

Listing 8: Aplicación de funciones en Java

Operadores

- En Idris, se pueden definir operadores, que no son más que funciones pero que se pueden aplicar de manera **infija** o **prefija**.
- Ejemplos de estos operadores son las funciones aritméticas que hemos visto con anterioridad:

```
x : Int
x = 10
y : Int
y = 20

z : Int
z = (+) 10 20

z' : Int
z = 10 + 20
```

Listing 9: Aplicación de operadores