Sesión de laboratorio 01

Mauricio Esquivel Reyes

10 de agosto de 2018

Índice

1. Introducción

1.1. ¿Qué es Haskell?

Haskell es un lenguaje de programación multi-propósito puramente funcional con semántica no estricta y tipado fuerte. Haskell cuenta con inferencia de tipos, por lo cual no es necesario especificar el tipo de las variables o las funciones. La mayoría de los lenguajes son imperativos, en estos se especifica

los pasos que debe realizar el código. Los lenguajes funcionales trabajan de forma diferente. En lugar de realizar acciones en secuencia, evalua expresiones.

1.1.1. Ejemplo Java

```
int pot(int n, int m){
   int res = 1;
   for(int i = 0; i < m; i++){
      res *= n;
   }
}
```

1.1.2. Ejemplo Haskell

```
pot n 0 = 1
pot n m = n * (pot n (m-1))
```

1.2. Instalación

En la página de `https://www.haskell.org/downloads` podrán encontrar los paquetes de Haskell. Existen dos versiones Haskell Platform que incluye el manejador de paquetes, el compilador GHC y otras herramientas. O también pueden descargar el compilador GHC que tiene el interpretre ghci.

1.3. Tipos básicos

Haskell tiene definido los siguientes tipos:

- Bool
  - True
  - False
  - &&
  - ||
  - not
- Char
  - ++

- Int
- Integers
- Float
- Double
    - +
    - -
    - /
    - *
- Listas
    - !!
- Tuplas
    - fst
    - snd

Para conocer el tipo de una expresión en haskell solo se necesita hacer:

```
> :t 5
> :t "Hola mundo"
> :t False
> :t head
```

1.4. Variables de tipo

Pero ¿qué es esa a? Los tipos que acabamos de ver empiezan con letra mayúscula. La a es una variable de tipo, podemos pensarla como los genericos de otros lenguajes. Estas variables de tipo son más poderosas que los genericos, ya que nos permiten escribir funciones muy generales mientras no dependan del comportamiento especifico de los tipos. Estas funciones son llamadas polimórficas.

1.5. Clases de tipo

Las clases de tipos son una especie de interfaz que define algún tipo de comportamiento. Si un tipo es un miembro de una clase de tipos, significa que ese tipo soporta e implementa el comportamiento que define la clase de tipos. Los podriamos ver como interfaces de Java.

```
> :t (==)
(==) :: Eq a => a -> a -> Bool
```

Cualquier cosa antes del símbolo =>es una restricción de clase. Se lee: La función de igualdad toma dos parámetros que son del mismo tipo y devuelve un Bool. El tipo de estos dos parámetros debe ser miembro de la clase Eq.

1.5.1. Básicas

- Eq
- Ord
- Show
- Read
- Enum

1.6. Más allá

Esta es una introducción a haskell muy muy básica. Para seguir aprendiendo hay bastante material en `https://www.haskell.org/documentation`

2. Lógica

2.1. Sintaxis

Esta es la sintaxis de la Lógica Proposicional que utilizaremos.

$$PL ::=< ProposicinAtmica > |\neg PL|(PL \wedge PL)|(PL \vee PL)|(PL \rightarrow PL)$$

$$< ProposicinAtmica >::= \top|\bot| < VariableProposicional >$$

$$< VariableProposicional >::= v < Indice >$$

$$< Indice >::= [i|i \in \mathbb{N}]$$

2.2. Definición en Haskell

```
-- Tipo de dato indice
type Indice = Int

-- Tipo de dato fórmula
data PL = Top | Bot
              | Var Indice | Oneg PL
              | Oand PL PL | Oor PL PL
              | Oimp PL PL deriving (Eq, Show)
```

2.3. Funciones

2.3.1. Número de operadores

```
numOp :: PL -> Int
numOp Top = 0
numOp Bot = 0
numOp (Var x) = 0
numOp (Oneg p) = numOp p + 1
numOp (Oand p q) = numOp p + numOp q + 1
numOp (Oor p q) = numOp p + numOp q + 1
numOp (Oimp p q) = numOp p + numOp q + 1
```

2.3.2. Elimina implicaciones

```
quitaImp :: PL -> PL
quitaImp Top = Top
quitaImp Bot = Bot
quitaImp (Var x) = Var x
quitaImp (Oneg p) = Oneg $ quitaImp p
quitaImp (Oand p q) = Oand (quitaImp p) (quitaImp q)
quitaImp (Oor p q) = Oor (quitaImp p) (quitaImp q)
quitaImp (Oimp p q) = Oor (Oneg $ quitaImp p) (quitaImp q)
```

2.3.3. Número de operadores binarios

```
numObin :: PL -> PL
numObin Top = 0
numObin Bot = 0
numObin (Var x) = 0
numObin (Oneg p) = numObin p
```

```
numObin (Oand p q) = numObin p + numObin q + 1
numObin (Oor p q) = numObin p + numObin q + 1
numObin (Oimp p q) = numObin p + numObin q + 1
```

2.3.4. Forma Normal Negativa

```
toNNF :: PL -> PL
toNNF p = toNNF $ quitaImp p where
  toNNF (Oneg (Oand p q)) = toNNF $ Oor (Oneg $ toNNF p) (Oneg $ toNNF q)
  toNNF (Oneg (Oor p q)) = toNNF $ Oand (Oneg $ toNNF p) (Oneg $ toNNF q)
  toNNF (Oneg (Oneg p)) = toNNF p
  toNNF (Oand p q) = Oand (toNNF p) (toNNF q)
  toNNF (Oor p q) = Oor (toNNF p) (toNNF q)
  toNNF p = p
```