Tema 2

Introducción a la programación con Haskell

Dpto. Ciencias de la Computación e
Inteligencia Artificial
Universidad de Sevilla

Contenidos

1. Introducción
2. El sistema GHC
3. Operaciones básicas
4. Funciones en Haskell
5. Guiones Haskell

1 Introducción

2 El sistema GHC

3 Operaciones básicas

4 Funciones en Haskell

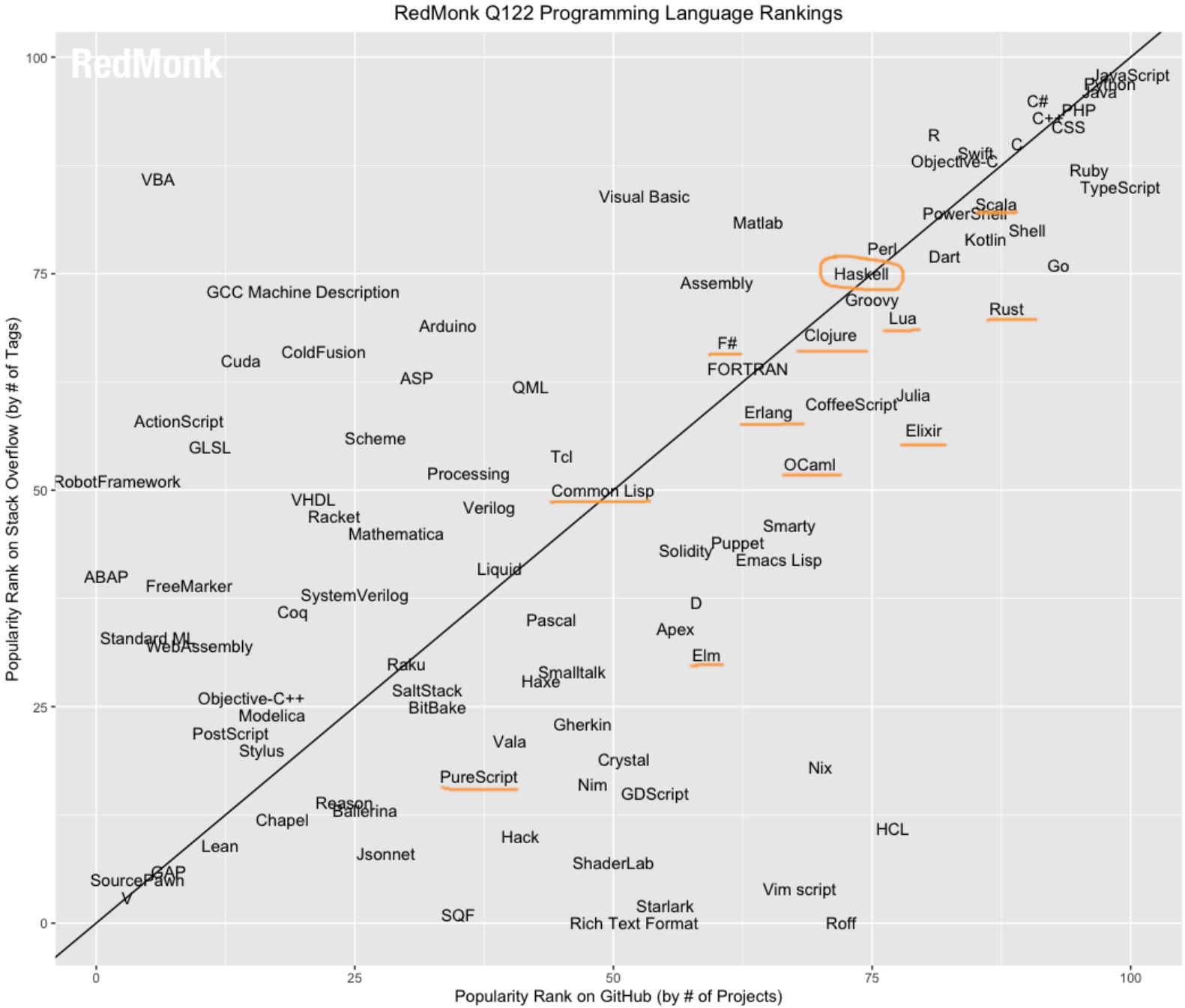
5 Guiones Haskell

¿Qué es Haskell?

- *Lenguaje de programación funcional perezoso*.
- Creado a finales de los años 80 por un comité ante la proliferación de lenguajes funcionales.
  - 1930s: Alonzo Church desarrolla el lambda cálculo (teoría básica de los lenguajes funcionales).
  - 1950s: John McCarthy desarrolla el Lisp (lenguaje funcional con asignaciones).
  - 1960s: Peter Landin desarrolla ISWIN (lenguaje funcional puro).
  - 1970s: John Backus desarrolla FP (lenguaje funcional con orden superior).
  - 1970s: Robin Milner desarrolla ML (lenguaje funcional con tipos polimórficos e inferencia de tipos).
  - 1980s: David Turner desarrolla Miranda (lenguaje funcional perezoso).
  - 1987: Un comité comienza el desarrollo de Haskell.
  - 2003: El comité publica el “Haskell Report"

¿Quién usa Haskell?

- Alcatel-Lucent: prototype narrowband software radio systems
- Facebook: manipulating PHP code base and fight spam
- Google: internal IT infrastructure support
- Microsoft: production serialization system and in Bond project (for schematized data)
- NVIDIA: in-house tools and backend development of its GPUs
- Fuente
- También: https://wiki.haskell.org/Haskell_in_industry


RedMonk Q122 Programming Language Rankings
RedMonk
Popularity Rank on Stack Overflow (by # of Tags)
Popularity Rank on GitHub (by # of Projects)
0
25
50
75
100
JavaScript
Python
Java
C#
PHP
C++
CSS
R
C
Swift
Objective-C
Ruby
TypeScript
Scala
PowerShell
Shell
Kotlin
Perl
Dart
Go
Haskell
Groovy
Rust
Lua
Clojure
F#
FORTRAN
Julia
CoffeeScript
Erlang
Elixir
OCaml
Common Lisp
Smarty
Puppet
Solidity
Emacs Lisp
D
Apex
Elm
Pascal
Smalltalk
Haxe
Raku
SaltStack
BitBake
Gherkin
Vala
Crystal
Nix
Nim
GDScript
PureScript
HCL
Hack
ShaderLab
Vim script
Starlark
Rich Text Format
Roff
SQF
Jsonnet
Lean
Chapel
Reason
Ballerina
GAP
SourcePawn
V
Objective-C++
Modelica
PostScript
Stylus
Liquid
Verilog
Mathematica
Racket
VHDL
Processing
Tcl
QML
ASP
Arduino
GCC Machine Description
Cuda
ColdFusion
Assembly
Visual Basic
Matlab
VBA
ActionScript
GLSL
Scheme
RobotFramework
ABAP
FreeMarker
SystemVerilog
Coq
Standard ML
WebAssembly

Rasgos característicos

- Haskell es un lenguaje funcional.
  - Programas concisos.
  - Sistema potente de tipos.
  - Creación de tipos de datos algebraicos.
  - Listas por comprensión.
  - Funciones recursivas.
  - Funciones de orden superior.
  - Evaluación perezosa.

Se recomienda leer este artículo, Haskell: el Lenguaje Funcional, para comprender mejor muchas de las bondades que acompañan a este lenguaje.

Además...

- Transparencia referencial
  - Son definiciones, no asignaciones
- Sin efectos secundarios
  - Como modificar el valor de una variable, ...
- Determinista
  - Calcular el valor de una función sobre unos argumentos siempre produce el mismo resultado.

Es lo que suele denominarse como programación funcional pura.

Perezoso

Una expresión no se calcula hasta que no es necesario.

- Permite trabajar con conjuntos infinitos de datos

```
λ> head []
*** Exception: Prelude.head: empty list
λ> map head [[1], []]
[1,*** Exception: Prelude.head: empty list
λ> head (map head [[1], []])
1
```

Se recomienda leer Algo pasa con Haskell, que además de detallar algunas características del lenguaje también menciona algunas limitaciones prácticas en el mundo del desarrollo actual.

Y por último...

- Tipado estático: Todas las expresiones tienen un tipo

```
sumaTriples :: [Int] -> Int
sumaTriples lst = sum (map (3*) lst)
```

```
λ> :t sum
sum :: (Num a, Foldable t) => t a -> a
λ> :t map
map :: (a -> b) -> [a] -> [b]
λ> :t (3*)
(3*) :: Num a => a -> a
```

1 Introducción

2 El sistema GHC

3 Operaciones básicas

4 Funciones en Haskell

5 Guiones Haskell

El sistema GHC

- GHC (Glasgow Haskell Compiler interpreter) es un compilador para generar binarios ejecutables.
- GHCi (Glasgow Haskell Compiler interpreter) es el intérprete de Haskell que usaremos en el curso.
  - Inicio mediante ghci:

```
c:\...> ghci
GHCi, version 8.4.3: http://www.haskell.org/ghc/ :? for help
Prelude>
```

  - `Prelude>` son las funciones de la librería por defecto de Haskell.

1 Introducción

2 El sistema GHC

3 Operaciones básicas

4 Funciones en Haskell

5 Guiones Haskell

Cálculo aritmético: Operaciones aritméticas

- Operaciones aritméticas en Haskell:

```
Prelude> 2+3
5
Prelude> 2-3
-1
Prelude> 2*3
6
Prelude> 7 `div` 2
3
Prelude> 2^3
8
```

Cálculo aritmético: Operaciones aritméticas

- Precedencia:

```
Prelude> 2*10^3
2000
Prelude> 2+3*4
14
```

- Asociatividad:

```
Prelude> 2^3^4
241785163922925834941235 2
Prelude> 2^(3^4)
241785163922925834941235 2
Prelude> 2-3-4
-5
Prelude> (2-3)-4
-5
```

Cálculo con listas: Seleccionar y eliminar

- Las colecciones básicas en Haskell son **listas**.
- Seleccionar el primer elemento de una lista no vacía:
  `head [1,2,3,4,5]` $\rightarrow$ `1`
- Eliminar el primer elemento de una lista no vacía:
  `tail [1,2,3,4,5]` $\rightarrow$ `[2,3,4,5]`
- Seleccionar el n–ésimo elemento de una lista (empezando en 0):
  `[1,2,3,4,5] !! 2` $\rightarrow$ `3`
- Seleccionar los n primeros elementos de una lista:
  `take 3 [1,2,3,4,5]` $\rightarrow$ `[1,2,3]`
- Eliminar los n primeros elementos de una lista:
  `drop 3 [1,2,3,4,5]` $\rightarrow$ `[4,5]`

Cálculo con listas

- Calcular la longitud de una lista:
  `length [1,2,3,4,5]` $\rightarrow$ `5`
- Calcular la suma de una lista de números:
  `sum [1,2,3,4,5]` $\rightarrow$ `15`
- Calcular el producto de una lista de números:
  `product [1,2,3,4,5]` $\rightarrow$ `120`
- Concatenar dos listas:
  `[1,2,3] ++ [4,5]` $\rightarrow$ `[1,2,3,4,5]`
- Invertir una lista:
  `reverse [1,2,3,4,5]` $\rightarrow$ `[5,4,3,2,1]`

Ejemplos de cálculos con errores

```
Prelude> 1 `div` 0
*** Exception: divide by zero
Prelude> head []
*** Exception: Prelude.head: empty list
Prelude> tail []
*** Exception: Prelude.tail: empty list
Prelude> [2,3] !! 5
*** Exception: Prelude.(!!): index too large
```

1 Introducción

2 El sistema GHC

3 Operaciones básicas

4 Funciones en Haskell

5 Guiones Haskell

Funciones: matemáticas *vs* haskell

- Notación para funciones en matemáticas:
  - En matemáticas, la aplicación de funciones se representa usando paréntesis y la multiplicación usando yuxtaposición o espacios.
  - Ejemplo:

    $$f(a, b) + cd$$

    representa la suma del valor de f aplicado a a y b más el producto de c por d.
- Notación para funciones en Haskell:
  - En Haskell, la aplicación de funciones se representa usando espacios y la multiplicación usando *.
  - Ejemplo: `f a b + c*d` representa la suma del valor de `f` aplicado a `a` y `b` más el producto de `c` por `d`.

Prioridad de la aplicación de funciones

- En Haskell, la aplicación de funciones tienen máxima prioridad de aplicación frente al resto de operadores. Por ejemplo, la expresión Haskell `f a + b` representa la expresión matemática $f(a) + b$.
- Ejemplos de expresiones Haskell y matemáticas:

Matemáticas	Haskell
$f(x)$	`f x`
$f(x, y)$	`f x y`
$f(g(x)) = f \circ g(x)$	`f (g x) = f . g x`
$f(x, g(y))$	`f x (g y)`
$f(x)g(y)$	`f x * g y`

- ¿Que haría si le damos `f g x` ?

Funciones en Haskell

- En Haskell, una **función** es una **aplicación** que toma uno o más **argumentos** y devuelve un **valor**[1].
- En Haskell, las funciones se definen mediante **ecuaciones** formadas por el **nombre de la función**, los **nombres de los argumentos** y el **cuerpo** que especifica cómo se calcula el valor a partir de los argumentos.
- Ejemplo de definición de función en Haskell:

```
doble x = x + x
```

- Ejemplo de evaluación:

```
doble 3
= 3 + 3      [def. de doble]
= 6          [def. de +]
```

[1] Matizaremos esto más adelante. Por el momento conviene considerarlo así

Nombres de funciones

- Los nombres de funciones tienen que empezar por una letra en minúscula. Por ejemplo, `sumaCuadrado`, `suma_cuadrado`, `suma'`
- Las palabras reservadas de Haskell no pueden usarse en los nombres de funciones. Algunas palabras reservadas son:

```
case class data   default deriving do   else
if   import in    infix   infixl   infixr instance
let  module newtype of    then     type   where
```

- Por convención, se acostumbra escribir los argumentos que son listas usando `s` como sufijo de su nombre. Por ejemplo:
  - `ns` representa una lista de números,
  - `xs` representa una lista de elementos,
  - `css` representa una lista de listas de caracteres.

La regla del sangrado

- En Haskell la disposición del texto del programa (el sangrado) delimita las definiciones mediante la siguiente regla:
  Una definición acaba con el primer trozo de código con un margen izquierdo menor o igual que el del comienzo de la definición actual.
- Ejemplo:

```
a = b + c
      where
            b = 1
            c = 2
d = a * 2
```

- Consejos:
  - Comenzar las definiciones de las funciones en la primera columna.
  - Usar el tabulador del editor de código para determinar el sangrado en las definiciones.

Comentarios en Haskell

- En los guiones o archivos de código de Haskell pueden incluirse comentarios.
- Un comentario simple comienza con – y se extiende hasta el final de la línea.
- Ejemplo de comentario simple:

```
-- (factorial n) es el factorial del número n.
factorial n = product [1..n]
```

- Un comentario multilínea comienza con {- y termina en -}
- Ejemplo de comentario multilínea:

```
{- (factorial n) es el factorial del número n.
Por ejemplo, factorial 3 == 6 -}
factorial n = product [1..n]
```

1 Introducción

2 El sistema GHC

3 Operaciones básicas

4 Funciones en Haskell

5 Guiones Haskell

Guiones Haskell

- En Haskell los usuarios pueden definir funciones.
- Las nuevas definiciones se definen en guiones, que son ficheros de textos compuestos por una sucesión de definiciones.
- Se acostumbra a identificar los ficheros de guiones de Haskell mediante la extensión `.hs`

Tu primer guión

- Se acostumbra a identificar los ficheros de guiones de Haskell mediante la extensión `.hs`
- Escribir en el guión las siguientes definiciones con un editor de código:

```
doble x = x+x
cuadruple x = doble (doble x)
```

- Cargar el guión en Haskell (ya graba):

```
Prelude> :l codigo.hs
```

- Evaluar ejemplos:

```
Prelude> cuadruple 10
40
Prelude> take (doble 2) [1,2,3,4,5,6]
[1,2,3,4]
```

El primer guión Haskell en Emacs

- Añadir al guión las siguientes definiciones:

```
factorial n = product [1..n]
media ns = sum ns `div` length ns
```

- Guardar y cargar el guión en Haskell.
- Evaluar ejemplos:

```
Prelude> factorial (doble 2)
24
Prelude> doble (media [1,5,3])
6
```

Bibliografía I

R. Bird. *Introducción a la programación funcional con Haskell*. Prentice Hall, 2000.

Capítulo 1: Conceptos fundamentales

G. Hutton *Programming in Haskell*. Cambridge University Press, 2007.

Chapter 2: First steps

B. O'Sullivan, D. Stewart y J. Goerzen. *Real World Haskell*. O'Reilly, 2008.

Chapter 1: Getting Started

B.C. Ruiz, F. Gutiérrez, P. Guerrero y J.E. Gallardo. Razonando con Haskell. Thompson, 2004..

Capítulo 2: Introducción a Haskell

S. Thompson. Haskell: The Craft of Functional Programming, Second Edition. Addison-Wesley, 1999.

Chapter 2: Getting started with Haskell and Hugs