Helium, Hugs y GHCi

Francisco José Martínez López

Salvador Léon Gil

Patricia Martín Cárdenas

Índice

Índice

Introducción

- **Haskell 98**

  Haskell nace en el año 1990 y su nombre se debe al matematico Haskell Curry.

  Haskell 98 es la versión *semi-oficial* de Haskell. Especifica una versión mínima compatible del lenguaje como base para futuras extensiones.

  Haskell soporta de forma natural: listas, patrones, recurrencia.

Introducción

- **Hugs**

  Es un intérprete portable y pequeño escrito en C.

  Compilación rápido, soporta compilación incremental.

  Al ser intérprete, no alcanza el tiempo de ejecución de, por ejemplo, GHC, nhc98, ó HBC.

  Hugs 98 es compatible con Haskell 98.

  Multiplataforma

  Interfaz gráfica: WinHugs.

Introducción

- **GHC**
  - Sus siglas “Glasgow Haskell Compiler”.
  - Compilador optimizado para Haskell.
  - Escrito en Haskell.
  - Menos portable que Hugs.
  - El compilador es más lento y consume más memoria que HUGS

Introducción

- **GHC**
  - El programa generado es más eficiente.
  - Soporta programación paralela y concurrente.
  - Software de memoria transaccional.
  - Multiplataforma
  - Tiene un interprete llamado GHCi.

Introducción

- **Helium**

  Helium es un lenguaje de programación funcional basado en Haskell98 (es un subconjunto).

  Es un compilador.

  Hint es su intérprete.

  Diseñado para la enseñanza de Haskell.

  La ausencia de sobrecarga es la más notable diferencia con Haskell.

Introducción

- **Helium**

  Los mensajes de error han sido mejorados.

  El compilador mantiene la pista de un montón de información para producir mensajes de errores informativos.

  Al no incluir clases los mensajes de error son más precisos.

Introducción

- **Otros compiladores:**

  NHC98

  Escrito en Haskell.

  Cumple el estandar Haskell98.

  Optimiza el ahorro de espacio en disco.

  Disponible en varias plataformas.

Introducción

- **Otros compiladores:**

  YHC

  Yhc es un derivado de nhc98, con los objetivos de ser más simple, más portable, más eficiente.

  JHC

  Jhc es un compilador experimental con el objetivo de probar nuevos métodos de optimización y explorar el diseño de las implementaciones Haskell.

Introducción

- **Otros compiladores:**

  HBI and HBC

  HBI es un interprete que puede cargar codigo compilado con el HBC.

  Implementa Haskell98.

  Este proyecto lleva bastante tiempo parado.

GHC vs Hugs

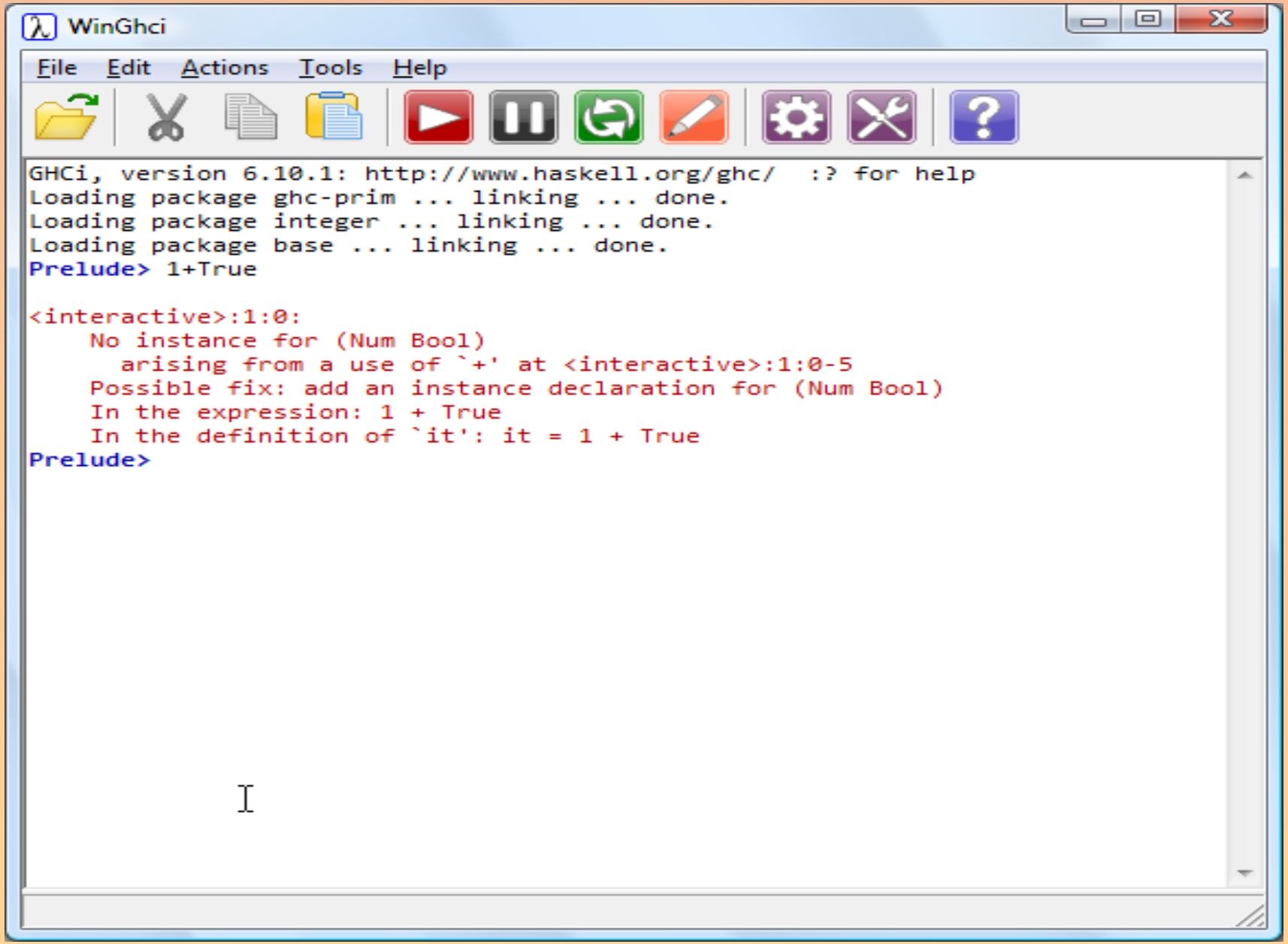
GHC

- Puede importar módulos de forma recursiva.
- Soporta ejecutar prgramas de forma paralela (SMP).
- Soporta concurrencia con sólo añadir un módulo.
- Hace falta tener el GHC para ejecutar un programa.

HUGS

- Como intérprete es más rápido.
- Se hace público un programa. El código es visible a todos.

WinGHCi


WinGhci
File Edit Actions Tools Help
GHCi, version 6.10.1: http://www.haskell.org/ghc/ :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer ... linking ... done.
Loading package base ... linking ... done.
Prelude> 1+True
<interactive>:1:0:
No instance for (Num Bool)
arising from a use of `+' at <interactive>:1:0-5
Possible fix: add an instance declaration for (Num Bool)
In the expression: 1 + True
In the definition of `it': it = 1 + True
Prelude>

Helium

- Instalación en sistemas Unix

  Descargamos los archivos fuente de la última versión de Helium (1.7)

  Descargamos la versión GHC6.8.2 (mínimo)

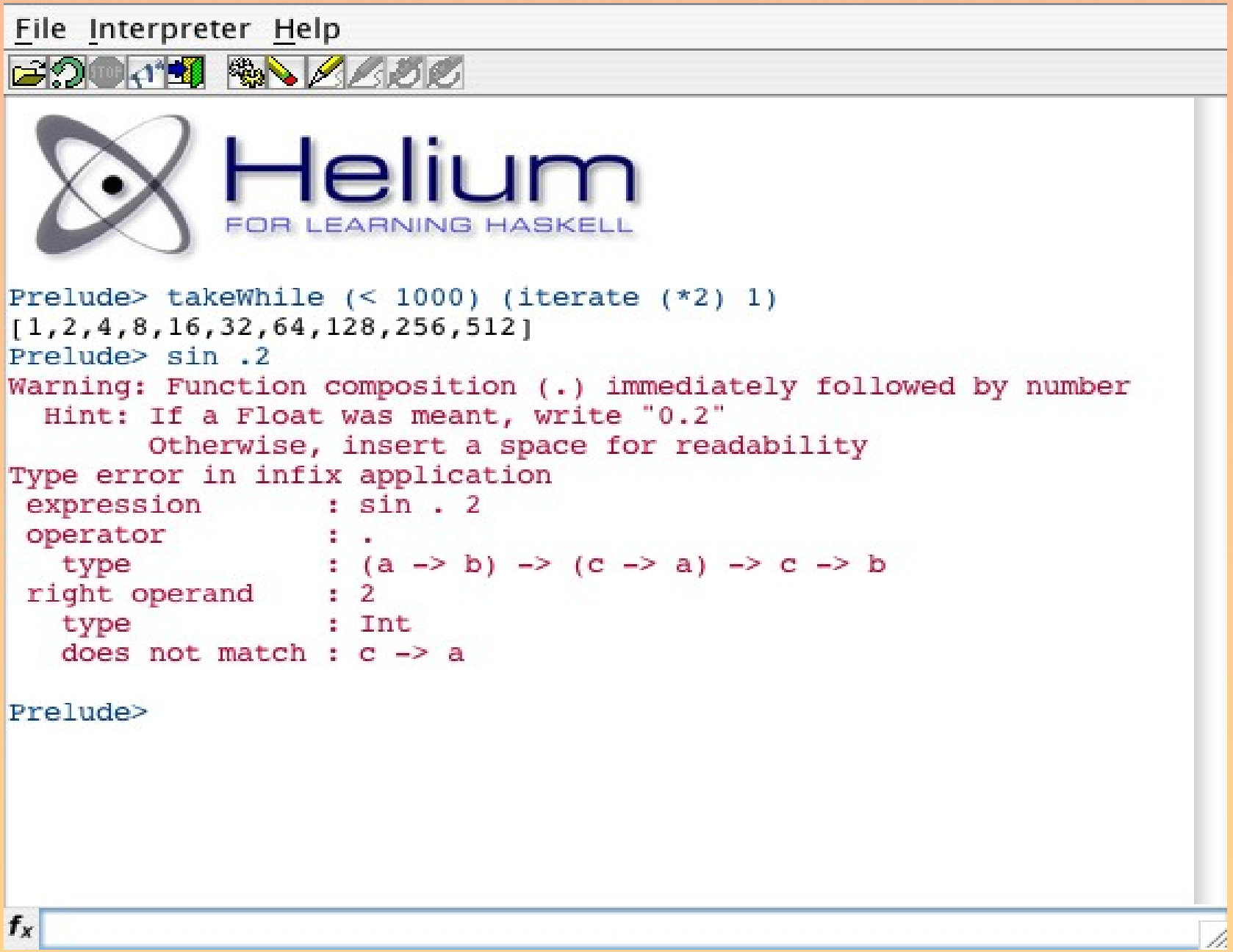
  Seguimos los pasos del fichero README de helium.

  En algunos casos hay que modificar el path para que encuentre el ejecutable de helium.

- Instalación en Windows

  Descargar el fichero ejecutable .exe

Helium


File Interpreter Help
Helium
FOR LEARNING HASKELL
Prelude> takeWhile (< 1000) (iterate (*2) 1)
[1,2,4,8,16,32,64,128,256,512]
Prelude> sin .2
Warning: Function composition (.) immediately followed by number
Hint: If a Float was meant, write "0.2"
Otherwise, insert a space for readability
Type error in infix application
expression : sin . 2
operator : .
type : (a -> b) -> (c -> a) -> c -> b
right operand : 2
type : Int
does not match : c -> a
Prelude>
fx

Helium

- Fue creado por:
  - Arjan van IJzendoorn, Rijk Jan van Haaften, Bastiaan Heeren and Daan Leijen
- Fue desarrollado en Utrecht University (Holanda)

Helium

- El lenguaje:
  - Es un subconjunto de Haskell
  - No soporta la definicion de clases e instancias.
  - El programador puede activar o desactivar la sobrecarga.
- El compilador:
  - Guarda guarda mucha informacion para poducir mensajes informativos.
    - Señla la localizacion exacta del origen (linea y columna).
  - Muchos mensajes de error están acompañados de pistas para su correcta resolución.

Helium

- El interprete
  - Hint es el interprete, basado en Java.
  - Se puede integrar con tu editor de texto favorito.
  - Puedes saltar a la localizacion exacta del error con solo un clik.
  - Muestra los tipos alineados en los mensajes de error, para facilitar la comparacion.
  - En caso de escribir el nombre de una funcion mal, te sugiere funciones con nombre parecido.

```
module Features2 where

main = concatmap (\x -> [f y]) "helium”

(3,8): Undefined variable "concatmap"

  Hint: Did you mean "concatMap" ?
```

Helium

- Warnigs:
  - Mas de un error puede ser mostrado cada vez.
  - El interprete avisa de variables sin usar, puesto que puede causar problemas.
  - Advierte de la ausencia de declaracion de tipos en funciones.
  - Un warning genera el tipo y se puede copiar y pegar en tu funcion.

```
(8,1): Warning: Missing type signature:
myFIlter :: (a -> Bool) -> [a] -> [a]
```

Helium

- No Soporta:
  - Declaraciones newtype
  - Asignar alias a campo de registro.
  - Renombrar e importar módulos con as.
  - Importar y exportar listas.
  - Clases, instancias y declaraciones por defecto.
  - Anotaciones estrictas.
  - N+k patrones.
  - (,) y (,,) ..para construir una tupla (tipo).
  - [], Como constructor de tipo.
  - Literate programming

Helium

- Soporta pero con restricciones:
  - Hay 5 constructores de tipos con las siguientes instancias:
    - Num: Int, Float
    - Eq: Bool, Char, Either a b, Float, Int, Maybe a, [a] y tuplas.
    - Ord: Bool, Char, Float, Int, [a] y tuplas
    - Show: Bool, Char, Either a b, Float, Int, Maybe a, Ordering, [a] y tuplas.
    - Enum: Bool, Char, Float, Int y ().
  - Los tipos de datos pueden derivar Show y Eq.
  - Si la funcion main no es de tipo IO, entonces el valor es mostrado con una funcion show.

Helium

- Soporta pero con restricciones:
  - El modulo es simple. Pero con suficiente potencia para los propositos de Helium:
  - Todo se exporta siempre: tipos de datos, funciones, sinonimos, instancias…
  - Todo lo que es importado es exportado.
  - No esta permitido importar algo que este presente en dos módulos distintos.
  - Puedes ocultar funciones en una declaracion importada con Hiding.
  - Prelude siempre es importado
    - Import Prelude hiding(map,filter)

Helium

- Soporta pero con restricciones:
  - Los literales numericos no estan sobrecargados (excepto cuando se usa –overloading flag)
  - Existe una sintaxis mas restrictiva para los operadores.
  - Tuplas con mas de 10 elementos no están permitidas.
  - \\, \n, \a, \b, \f, \r, \t, \v, \", \'. Son soportados en caracteres y Strings
  - No esta permitida secuencias como \030.

Helium

- En Helium pero no en Haskell
  - Una función show es generada para cada tipo de datos y tipo sinónimo
  - La función showMaybe se crea automaticamente para el tipo de datos Maybe.
  - Si el tipo de datos tiene parámetros entonces la función show toma los argumentos adicionales necesarios.

Mensajes de error en los distintos compiladores/interpretes

Errores

Código:

```
exOr :: bool -> bool -> bool

exOr b1 b2 = (b1 && not b2) || (b2 && not b1)
```

GHC

```
Couldn't match expected type 'Bool' against inferred
    type 'bool' (a rigid variable) 'bool' is bound by the
    type signature for 'exOr' at cap03.hs:3:8

In the first argument of '(&&)' , namely 'b1'

In the first argument of '(||)' ,namely '(b1&&(notb2))'

In the expression: (b1&&(notb2))||(b2&&(notb1))
```

Errores

Código:

```
exOr :: bool -> bool -> bool

exOr b1 b2 = (b1 && not b2) || (b2 && not b1)
```

HUGS

```
ERROR "cap03.hs":5 -Inferred type is not general
   enough

***Expression:exOr

***Expected type: a->a->a

***Inferred type:Bool->Bool->Bool
```

Errores

Código:

```
exOr :: bool -> bool -> bool

exOr b1 b2 = (b1 && not b2) || (b2 && not b1)
```

HELIUM

```
(2,46): Syntax error:

    unexpected '='

    expecting type, '->', next in block (based on layout),
    ';' or end of block (based on layout)

Compilation failed with 1 error
```

Errores

Código

fun x

fun 2 = 34

GHC

parse error (possibly incorrect indentation)

HUGS

Syntax error indeclaration (unexpected ';', possibly due to bad layout)

Errores

Código

```
fun x

fun 2 = 34
```

HELIUM

```
(6,5): Warning: Variable "x" is not used

(6,7): Warning: Variable "fun" is not used

(6,7): Warning: Variable "fun" shadows the one at (6,1)

(6,1): Warning: Missing type signature: fun :: a -> b -> Int ->
    Int

(6,1): Warning: Missing pattern in function bindings:

  fun _ _ _ = ...

Compilation successful with 5 warnings
```

Errores

Código:

cos sin 3

GHC

No instance for (Floating(a->a))

arising from use of 'cos' at<interactive>:1:0-8

Possible fix: add an instance declaration for (Floating (a->a))

In the expression:cos sin 3

In the definition of 'it' : it=cos sin 3

Errores

Código:

cos sin 3

HUGS

ERROR - Cannot infer instance

*** Instance : Floating (a -> a)

*** Expression : cos sin 3

Errores

Código:

```
cos sin 3
```

HELIUM

```
Type error in application
 expression       : sin cos 3
 term             : sin
   type           : Float          -> Float
   does not match : (Float -> Float) -> Int -> a
 because          : too many arguments are given
```

Errores

Código:

```
fun x x = 17
```

GHC

```
Conflicting definitions for `x'
In the definition of 'fun'
```

HUGS

```
Repeated variable "x" in pattern
```

HELIUM

```
(14,5), (14,7): Duplicated variable "x"
Compilation failed with 1 error
```

Errores

Código:

```
f [] = 23

g y = 2

f (x:xs) = 6
```

GHC

```
Multiple declarations of `Main.f'

Declared at: cap03.hs:3:0

cap03.hs:7:0
```

Además, nos dice en qué líneas están las declaraciones.

Errores

Código:

```
f [] = 23
g y = 2
f (x:xs) = 6
```

HUGS

"f" multiply defined

Errores

Código:

```
f [] = 23

g y = 2

f (x:xs) = 6
```

HELIUM

```
(18,15): Syntax error:

    unexpected '='

    expecting expression, operator, '::', keyword
    'where', next in block (based on layout), ';' or end of
    block (based on layout)

Compilation failed with 1 error
```

Errores

Código:

```
foo :: Int -> [Char]
foo x = ['1'] ++ foo(x div 10)
```

GHC

```
Couldn't match expected type `(a -> a -> a) -> t -> Int'
against inferred type `Int'
In the first argument of `foo', namely `(x div 10)'
In the second argument of `(++)', namely `foo (x div
    10)'
In the expression: ['1'] ++ (foo (x div 10))
```

Errores

Código:

```
foo :: Int -> [Char]
foo x = ['1'] ++ foo(x div 10)
```

HUGS:

```
Type error in application
*** Expression : x div 10
*** Term : x
*** Type : Int
*** Does not match : a -> b -> c
```

Errores

Código:

```
foo :: Int -> [Char]

foo x = ['1'] ++ foo(x div 10)
```

HELIUM:

```
(22,28): Syntax error:

    unexpected '='

    expecting type, '->', next in block (based on layout),
    ';' or end of block (based on layout)

Compilation failed with 1 error
```

Errores

Código:

```
empty :: [a] -> Bool
empty as = (as == [])
```

GHC

```
No instance for (Eq a)
arising from use of `==' at cap03.hs:4:12-19
Possible fix: add (Eq a) to the type signature(s) for
    `empty'
In the expression: (as == [])
In the definition of `empty': empty as = (as == [])
```

Errores

Código:

```
empty :: [a] -> Bool
empty as = (as == [])
```

HUGS

```
Cannot justify constraints in explicitly typed binding
*** Expression : empty
*** Type : [a] -> Bool
*** Given context : ()
*** Constraints : Eq a
```

Errores

Código:

```
empty :: [a] -> Bool

empty as = (as == [])
```

HELIUM

```
(26,31): Syntax error:

    unexpected '='

    expecting type, '->', next in block (based on layout),
    ';' or end of block (based on layout)

Compilation failed with 1 error
```

Conclusiones

- **Mensajes**

  GHC: Muy bueno.

  HUGS: A veces un poco confuso.

  HELIUM: Muy bueno, lo mejor son las sugerencias.

- **Tamaño**

  GHC: El compilador es un archivo muy pesado (casi 100Mb).

  HUGS: Ligero, ocupa poco.

  HELIUM: Ligero, aunque requiere Java.

- **Herramientas**

  GHC: Gran variedad de herramientas y extensiones.

  HUGS: No muchas.

  HELIUM: No muchas.

Conclusiones

- **HUGS**: es el más rápido de los tres a la hora de interpretar. Muy útil en la docencia.
- **GHC**: Tiene disponibles muchas extensiones, además de hacer un código muy eficiente.
- **HELIUM**: Aún no soporta clases (gran error) que lo hace incompatible con mucho código de Haskell98.

Bibliografía

- Wikipedia
- http://shootout.alioth.debian.org/ [May 2009]
- http://www.haskell.org/ghc/index.html [May 2009]
- http://proyectoprincipia.wordpress.com/tag/haskell/ [Dic 2008]
- http://www.haskell.org/haskellwiki/Es/Implementacio
- http://www.cs.kent.ac.uk/people/staff/sjt/craft2e/error
- http://www.cs.uu.nl/wiki/Helium [Nov 2008]
- http://cvs.haskell.org/Hugs/pages/users_guide/index
- http://code.google.com/p/winghci/ [Mar 2009]

Preguntas

¿Preguntas?