

# $\lambda$ Page

*Fernando Benavides*

Departamento de Computación, FCEyN, Universidad de Buenos Aires.

16 de julio de 2010

# El Orador

*¿quién soy?*

- ▶ Fernando Benavides

*¿cómo llegué hasta aquí?*

- ▶ Alumno de Computación desde 2001
- ▶ Programador desde hace más de 10 años
- ▶ Programador *Funcional* desde hace 2 años

# El Orador

*¿quién soy?*

- ▶ Fernando Benavides

*¿cómo llegué hasta aquí?*

- ▶ Alumno de Computación desde 2001
- ▶ Programador desde hace más de 10 años
- ▶ Programador *Funcional* desde hace 2 años

# $\lambda$ Page

*Un bloc de notas para usuarios Haskell*

*Una herramienta para...*

- ▶ debuggear
- ▶ entender código
- ▶ realizar *micro-testing*

# *λPage*

*Un bloc de notas para usuarios Haskell*

*Una herramienta para...*

- ▶ debuggear
- ▶ entender código
- ▶ realizar *micro-testing*

# Herramientas del Programador Funcional

## *GHCi*

**Descripción** Es la consola de *GHC*

- Ventajas**
- ▶ Permite evaluar expresiones, verificar su tipo o su género
  - ▶ Permite cargar o importar módulos
  - ▶ Permite ejecutar acciones de entrada/salida

- Desventajas**
- ▶ No permite editar más de una expresión a la vez
  - ▶ Las opciones del compilador se setean una única vez al iniciar
  - ▶ Las definiciones se pierden al recargar módulos o cerrar el programa

# Herramientas del Programador Funcional

## *GHCi*

**Descripción** Es la consola de *GHC*

- Ventajas**
- ▶ Permite evaluar expresiones, verificar su tipo o su género
  - ▶ Permite cargar o importar módulos
  - ▶ Permite ejecutar acciones de entrada/salida

- Desventajas**
- ▶ No permite editar más de una expresión a la vez
  - ▶ Las opciones del compilador se setean una única vez al iniciar
  - ▶ Las definiciones se pierden al recargar módulos o cerrar el programa

# Herramientas del Programador Funcional

## *Hugs98*

**Descripción** Es un intérprete *Haskell* escrito en *C*

- Ventajas**
- ▶ Es pequeño y portable
  - ▶ Ideal para aprender *Haskell*
  - ▶ Provee una interfaz gráfica para *Windows* (*WinHugs*)

- Desventajas**
- ▶ Bajo rendimiento en tiempo de ejecución
  - ▶ No permite compilar código



# Herramientas del Programador Funcional

## *Hugs98*

**Descripción** Es un intérprete *Haskell* escrito en *C*

- Ventajas**
- ▶ Es pequeño y portable
  - ▶ Ideal para aprender *Haskell*
  - ▶ Provee una interfaz gráfica para *Windows* (*WinHugs*)

- Desventajas**
- ▶ Bajo rendimiento en tiempo de ejecución
  - ▶ No permite compilar código

# Herramientas del Programador Funcional

## *Hat*

**Descripción** Herramienta para realizar seguimiento a través de trazas

**Ventajas**

- ▶ Ayuda a localizar errores en los programas
- ▶ Permite entender el funcionamiento de los mismos

**Desventajas**

- ▶ Requiere que el programa compile para poder ejecutarse
- ▶ Requiere ejecutar un programa antes de poder analizarlo
- ▶ Se encuentra sin mantenimiento activo

# Herramientas del Programador Funcional

## *Hat*

**Descripción** Herramienta para realizar seguimiento a través de trazas

**Ventajas**

- ▶ Ayuda a localizar errores en los programas
- ▶ Permite entender el funcionamiento de los mismos

**Desventajas**

- ▶ Requiere que el programa compile para poder ejecutarse
- ▶ Requiere ejecutar un programa antes de poder analizarlo
- ▶ Se encuentra sin mantenimiento activo

# Necesidades

Quienes programan en lenguajes orientados a objetos utilizan herramientas como

- ▶ **Java** Scrapbook Pages
- ▶ Workspace de **Smalltalk**

Estas aplicaciones permiten

- ▶ introducir pequeñas porciones de código para ejecutarlas, inspeccionarlas y analizar los resultados obtenidos
- ▶ administrar varias paginas de texto en las que dichas expresiones pueden intercalarse con texto libre
- ▶ crear objetos localmente y utilizarlos

# Necesidades

Quienes programan en lenguajes orientados a objetos utilizan herramientas como

- ▶ **Java** Scrapbook Pages
- ▶ Workspace de **Smalltalk**

Estas aplicaciones permiten

- ▶ introducir pequeñas porciones de código para ejecutarlas, inspeccionarlas y analizar los resultados obtenidos
- ▶ administrar varias paginas de texto en las que dichas expresiones pueden intercalarse con texto libre
- ▶ crear objetos localmente y utilizarlos

# *λPage*

*λPage* es una herramienta similar al Workspace de Smalltalk, en tanto:

- ▶ permite al desarrollador trabajar con texto libre
- ▶ detecta expresiones y definiciones válidas
- ▶ permite inspeccionarlas y evaluarlas

Además, *λPage*, brinda otras facilidades particulares para *Haskell*:

- ▶ Integración con *Cabal* y *Hayoo*!
- ▶ Aprovecha *lazy evaluation* y *tipado estático*
- ▶ Presenta resultados dinámicamente

# *λPage*

*λPage* es una herramienta similar al Workspace de Smalltalk, en tanto:

- ▶ permite al desarrollador trabajar con texto libre
- ▶ detecta expresiones y definiciones válidas
- ▶ permite inspeccionarlas y evaluarlas

Además, *λPage*, brinda otras facilidades particulares para *Haskell*:

- ▶ Integración con *Cabal* y *Hayoo*!
- ▶ Aprovecha *lazy evaluation* y *tipado estático*
- ▶ Presenta resultados dinámicamente

# Interpretación de Expresiones

*λPage* permite interpretar expresiones como:

```
[1 , 2 , 3]
```

```
xs = [1 , 2 , 3] :: [Float]  
map (+) xs
```

```
[1 , 2 ..]
```

```
let loop = loop in 1:loop
```



# Interpretación de Expresiones

*λPage* permite interpretar expresiones como:

```
[1 , 2 , 3]
```

```
xs = [1 , 2 , 3]  ::  [ Float ]  
map (+) xs
```

```
[1 , 2 .. ]
```

```
let loop = loop in 1:loop
```

# Interpretación de Expresiones

*λPage* permite interpretar expresiones como:

```
[1 , 2 , 3]
```

```
xs = [1 , 2 , 3]  ::  [ Float ]  
map (+) xs
```

```
[1 , 2 ..]
```

```
let loop = loop in 1:loop
```

# Interpretación de Expresiones

*λPage* permite interpretar expresiones como:

```
[1 , 2 , 3]
```

```
xs = [1 , 2 , 3]  ::  [ Float ]  
map (+) xs
```

```
[1 , 2 ..]
```

```
let loop = loop in 1:loop
```

# Presentación de Resultados

- ▶ Para expresiones inválidas,  $\lambda$ Page presenta el error informado por *GHC*
- ▶ Para expresiones sin resultado “visible”,  $\lambda$ Page permite conocer su tipo
- ▶ Para expresiones infinitas,  $\lambda$ Page presenta su valor incrementalmente hasta que el usuario cancela la evaluación
- ▶ Para expresiones que requieren cálculos infinitos,  $\lambda$ Page permite al usuario cancelar la evaluación

## Acciones con Efectos Colaterales

Para realizar acciones que puedan generar efectos colaterales, en *Haskell* se utiliza la mónada *IO*. Por ejemplo:

```
readFile 'README' :: IO String
```

*IO String* no es un tipo “visible”, por lo que el valor de la expresión no se podría mostrar.

Pero *λPage* toma el modelo de *GHCi* y ejecuta la acción, presentando su resultado

## Acciones con Efectos Colaterales

Para realizar acciones que puedan generar efectos colaterales, en *Haskell* se utiliza la mónada *IO*. Por ejemplo:

```
readFile 'README' :: IO String
```

*IO String* no es un tipo “visible”, por lo que el valor de la expresión no se podría mostrar.

Pero *λPage* toma el modelo de *GHCi* y ejecuta la acción, presentando su resultado

## Acciones con Efectos Colaterales

Para realizar acciones que puedan generar efectos colaterales, en *Haskell* se utiliza la mónada *IO*. Por ejemplo:

```
readFile 'README' :: IO String
```

*IO String* no es un tipo “visible”, por lo que el valor de la expresión no se podría mostrar.

Pero *λPage* toma el modelo de *GHCi* y ejecuta la acción, presentando su resultado

# Listas

*λPage* interpreta de modo particular las listas (expresiones de tipo `Show a => [a]`)

Tomemos como ejemplo la siguiente expresión:

```
let loop = loop in [1, div 0 0, 2,  
                     undefined, 3, loop, 4]
```



## Listas

```
let loop = loop in [1, div 0 0, 2,  
                    undefined, 3, loop, 4]
```

Si *λPage* presentase su evaluación tal como lo hace con las demás expresiones, el resultado sería

```
[1,
```

e informaría al usuario la excepción encontrada (o sea, `DivideByZero`)

## Listas

```
let loop = loop in [1, div 0 0, 2,  
                    undefined, 3, loop, 4]
```

*λPage* en cambio, podría evaluar cada elemento por separado y detectar excepciones. En tal caso, el resultado sería

```
[1, ⊥, 2, ⊥, 3,
```

y continuaría intentando calcular el siguiente elemento hasta que el usuario decidiese cancelar

## Listas

```
let loop = loop in [1, div 0 0, 2,  
                    undefined , 3, loop , 4]
```

*λPage*, sin embargo, detecta cálculos posiblemente infinitos y presenta como resultado:

```
[1, ⊥, 2, ⊥, 3, ⊥, 4]
```

permitiendo luego al usuario conocer el motivo de cada  $\perp$  a través de un menú contextual

## Manejo de Módulos

Utilizando *λPage*, el usuario puede

- ▶ Importar módulos
- ▶ Cargar módulos
- ▶ Recargar módulos
- ▶ Ver los módulos de un paquete *Cabal*
- ▶ Ver la documentación de un módulo utilizando *Hayoo!*

Todo esto *sin perder las expresiones que ya tiene definidas*

# Paralelismo

Muchas cosas suceden al mismo tiempo en *λPage*

- ▶ Manejo de Páginas (crear, abrir, modificar, guardar, cerrar, etc.)
- ▶ Interpretación de Expresiones
- ▶ Evaluación de Acciones de Entrada/Salida

Para lograrlo, creamos *eprocess*:

- ▶ Basada conceptualmente en *Erlang*
- ▶ Construida utilizando *Threads*, *Channels* y *MVars*

# Paralelismo

Muchas cosas suceden al mismo tiempo en *λPage*

- ▶ Manejo de Páginas (crear, abrir, modificar, guardar, cerrar, etc.)
- ▶ Interpretación de Expresiones
- ▶ Evaluación de Acciones de Entrada/Salida

Para lograrlo, creamos *eprocess*:

- ▶ Basada conceptualmente en *Erlang*
- ▶ Construída utilizando *Threads*, *Channels* y *MVars*

# Principales Decisiones de Diseño

*λPage* es el fruto de un diseño evolutivo que atravesó las siguientes etapas:

- ▶ Conexión con *GHC* utilizando *hint*
- ▶ Paralelismo a través de *eprocess*
- ▶ Creación de **servers**
- ▶ Integración con *Cabal* y *Hayoo!*
- ▶ Presentación de Interpretaciones
  - ▶ Expresiones "visibles"
  - ▶ Expresiones "no visibles"
  - ▶ Entrada / Salida
  - ▶ Listas

# Principales Decisiones de Diseño

*λPage* es el fruto de un diseño evolutivo que atravesó las siguientes etapas:

- ▶ Conexión con *GHC* utilizando *hint*
- ▶ Paralelismo a través de *eprocess*
- ▶ Creación de **servers**
- ▶ Integración con *Cabal* y *Hayoo!*
- ▶ Presentación de Interpretaciones
  - ▶ Expresiones "visibles"
  - ▶ Expresiones "no visibles"
  - ▶ Entrada / Salida
  - ▶ Listas



# Principales Decisiones de Diseño

*λPage* es el fruto de un diseño evolutivo que atravesó las siguientes etapas:

- ▶ Conexión con *GHC* utilizando *hint*
- ▶ Paralelismo a través de *eprocess*
- ▶ Creación de **servers**
- ▶ Integración con *Cabal* y *Hayoo!*
- ▶ Presentación de Interpretaciones
  - ▶ Expresiones “visibles”
  - ▶ Expresiones “no visibles”
  - ▶ Entrada / Salida
  - ▶ Listas

# Principales Decisiones de Diseño

*λPage* es el fruto de un diseño evolutivo que atravesó las siguientes etapas:

- ▶ Conexión con *GHC* utilizando *hint*
- ▶ Paralelismo a través de *eprocess*
- ▶ Creación de **servers**
- ▶ Integración con *Cabal* y *Hayoo!*
- ▶ Presentación de Interpretaciones
  - ▶ Expresiones “visibles”
  - ▶ Expresiones “no visibles”
  - ▶ Entrada / Salida
  - ▶ Listas

# Principales Decisiones de Diseño

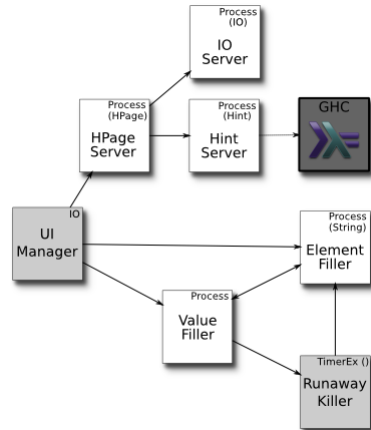
*λPage* es el fruto de un diseño evolutivo que atravesó las siguientes etapas:

- ▶ Conexión con *GHC* utilizando *hint*
- ▶ Paralelismo a través de *eprocess*
- ▶ Creación de **servers**
- ▶ Integración con *Cabal* y *Hayoo!*
- ▶ Presentación de Interpretaciones
  - ▶ Expresiones “visibles”
  - ▶ Expresiones “no visibles”
  - ▶ Entrada / Salida
  - ▶ Listas

# Arquitectura

## Principales Requerimientos:

- Conexión con GHC
- Paralelismo
- Errores Controlados
- Presentación de Resultados



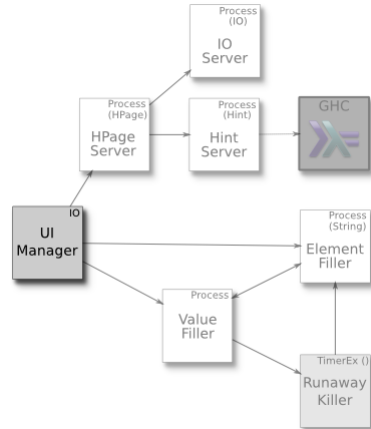
## Ejemplo de Interacción

Veremos cómo interactúan estos componentes para evaluar la siguiente expresión:

```
readFile "hpage.cabal" >>=  
  return . length . head . lines
```

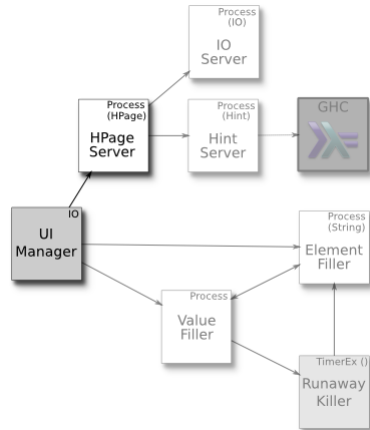
## Ejemplo de Interacción

El usuario indica al UI Manager que desea interpretar la expresión



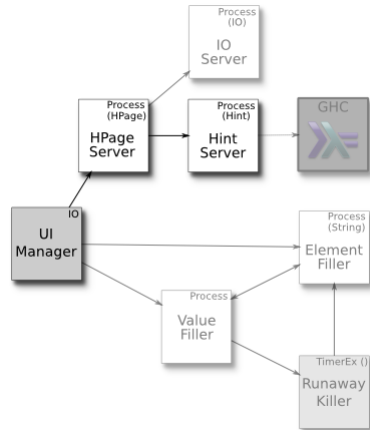
## Ejemplo de Interacción

El UI Manager solicita la evaluación del texto al HPage Server



## Ejemplo de Interacción

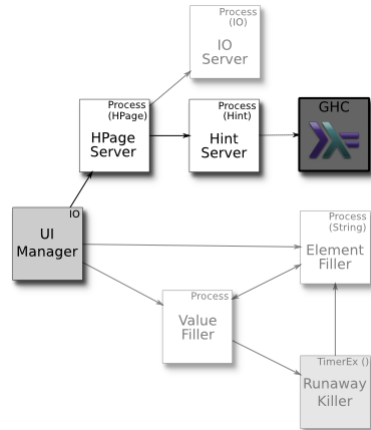
El HPage Server envía al Hint Server la expresión para conocer su valor y su tipo





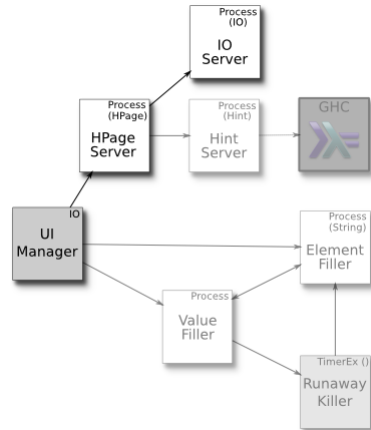
## Ejemplo de Interacción

El Hint Server se comunica con GHC utilizando *hint* y obtiene los valores solicitados, que luego informa al HPage Server



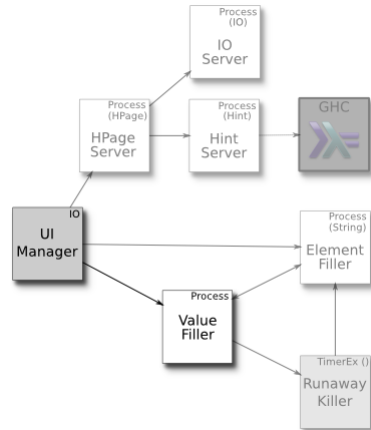
## Ejemplo de Interacción

El HPage Server, al detectar que se trata de una acción de la mónada IO, crea una MVar y envía un mensaje al IO Server para que ejecute la acción y llene la MVar con su resultado. Luego informa al UI Manager el tipo de la expresión y la MVar en la que se volcará su resultado una vez ejecutada



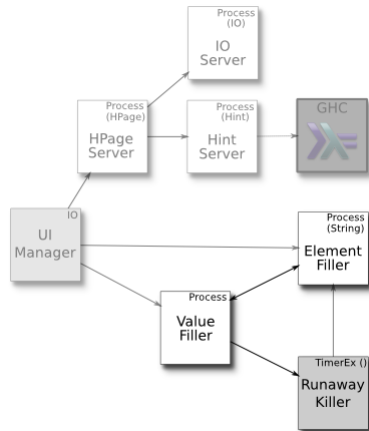
## Ejemplo de Interacción

El UI Manager muestra el tipo al usuario y vacía el cuadro de texto donde se encuentra el resultado. Luego, informándole la MVar recibida, solicita al Value Filler que espere el resultado y llene el cuadro de texto con él.



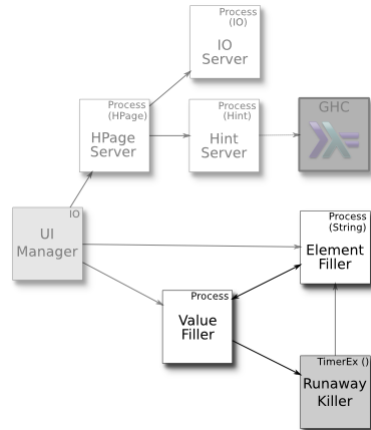
## Ejemplo de Interacción

El Value Filler se bloquea a la espera de que un valor sea depositado en la MVar recibida. Una vez conseguido ese valor, que es una cadena de caracteres ("11"), toma el primero y lo envía al Element Filler al tiempo que inicia el Runaway Killer. Luego, se bloquea esperando recibir un mensaje de alguno de ellos.



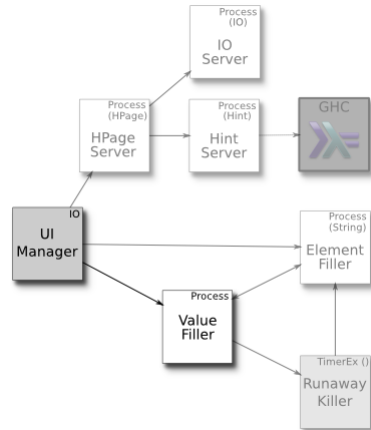
## Ejemplo de Interacción

Antes de transcurrido un segundo, el Element Filler computa el caracter 1 y envía un mensaje al Value Filler con el mismo. El Value Filler lo agrega al cuadro de texto del resultado y detiene el Runaway Killer. Luego repite el paso anterior y éste para el segundo caracter.



## Ejemplo de Interacción

Finalmente, el Value Filler informa al UI Manager que ha presentado el resultado con éxito, para que éste se lo informe al usuario.



## *eprocess*

```
newtype ReceiverT r m a
```

```
type Process r = ReceiverT r IO
```

```
spawn :: MonadIO m => Process r k -> m (Handle r)  
kill  :: MonadIO m => Handle a -> m ()  
self  :: Monad m => ReceiverT r m (Handle r)  
sendTo :: MonadIO m => Handle a -> a -> m ()  
recv  :: MonadIO m => ReceiverT r m r
```

## *eprocess*

```
newtype ReceiverT r m a
```

```
type Process r = ReceiverT r IO
```

```
spawn :: MonadIO m => Process r k -> m (Handle r)  
kill  :: MonadIO m => Handle a -> m ()  
self  :: Monad m => ReceiverT r m (Handle r)  
sendTo :: MonadIO m => Handle a -> a -> m ()  
recv  :: MonadIO m => ReceiverT r m r
```



## *eprocess*

```
newtype ReceiverT r m a
```

```
type Process r = ReceiverT r IO
```

```
spawn :: MonadIO m => Process r k -> m (Handle r)  
kill  :: MonadIO m => Handle a -> m ()  
self  :: Monad m => ReceiverT r m (Handle r)  
sendTo :: MonadIO m => Handle a -> a -> m ()  
recv  :: MonadIO m => ReceiverT r m r
```

## Servers (IOServer)

```
newtype ServerHandle = SH { handle :: Handle (IO ()) }

start :: IO ~ ServerHandle
start = spawn ioRunner >>= return . SH
    where ioRunner = forever $ recv >>= liftIO

runIn :: ServerHandle -> IO a ->
    IO (Either SomeException a)
runIn server action = runHere $ do
    me <- self
    sendTo (handle server) $ try action >>=
        sendTo me
        recv

stop :: ServerHandle -> IO ()
stop = kill . handle
```

## HPage.Control

```
newtype Expression = Exp {exprText :: String}  
    deriving (Eq, Show)  
  
data Page = Page { — Display —  
    expressions :: [Expression],  
    currentExpr :: Int,  
    undoActions :: [HPage ()],  
    redoActions :: [HPage ()],  
    — File System —  
    original :: [Expression],  
    filePath  :: Maybe FilePath  
}
```

## HPage.Control

```
data Context = Context { — Package —  
  activePackage :: Maybe PackageIdentifier ,  
  pkgModules   :: [Hint.ModuleName] ,  
  — Pages —  
  pages        :: [Page] ,  
  currentPage  :: Int ,  
  — GHC State —  
  loadedModules :: Set String ,  
  importedModules :: Set String ,  
  extraSrcDirs  :: [FilePath] ,  
  ghcOptions    :: String ,  
  server        :: HS.ServerHandle ,  
  ioServer      :: HPIO.ServerHandle ,  
  — Actions —  
  recoveryLog   :: Hint.InterpreterT IO () }
```

# Objetivos Alcanzados

*λPage* permite al usuario

- ▶ Configurar de entorno según paquetes *Cabal*
- ▶ Editar páginas mientras se evalúa una expresión
- ▶ Visualizar expresiones con errores o cálculos infinitos sin bloquearse mostrando el resultado más completo posible
- ▶ Importar, cargar y recargar módulos sin perder las expresiones con las que está trabajando
- ▶ Intercalar definiciones, expresiones y texto libre

## Logros Adicionales

Más allá de los objetivos propuestos para esta tesis, *λPage* permite

- ▶ Consultar la API de *Haskell* utilizando *Hayoo*!
- ▶ Trabajar en *OSX*, *Windows* y *Linux*
- ▶ Administrar varias páginas con expresiones, pudiendo cargarlas, guardarlas, etc.
- ▶ Determinar el contexto de evaluación dinámicamente
- ▶ Navegar módulos importados o cargados

# Tareas a Realizar

- ▶ Presentación de resultados:
  - ▶ Nuevas visualizaciones
  - ▶ Distintos tipos a tratar de modo particular
  - ▶ Composición
- ▶ Otras herramientas
  - ▶ Soporte para TDD
  - ▶ Refactoring
  - ▶ Análisis de Terminación
  - ▶ Debugging
- ▶ Otros lenguajes
  - ▶ Erlang

# Tareas a Realizar

- ▶ Presentación de resultados:
  - ▶ Nuevas visualizaciones
  - ▶ Distintos tipos a tratar de modo particular
  - ▶ Composición
- ▶ Otras herramientas
  - ▶ Soporte para TDD
  - ▶ Refactoring
  - ▶ Análisis de Terminación
  - ▶ Debugging
- ▶ Otros lenguajes
  - ▶ Erlang



# Tareas a Realizar

- ▶ Presentación de resultados:
  - ▶ Nuevas visualizaciones
  - ▶ Distintos tipos a tratar de modo particular
  - ▶ Composición
- ▶ Otras herramientas
  - ▶ Soporte para TDD
  - ▶ Refactoring
  - ▶ Análisis de Terminación
  - ▶ Debugging
- ▶ Otros lenguajes
  - ▶ Erlang

# ¡Gracias a todos!

- ▶ *Sitio Web de λPage:*
  - ▶ <http://hpage.haskell.com>
- ▶ *λPage en Github*
  - ▶ <http://github.com/elbrujohalcon/hPage>
- ▶ *Fernando Benavides en la Internet*
  - ▶ <http://profiles.google.com/greenmellon>