

$\lambda Page$

Fernando Benavides

Departamento de Computación, FCEyN, Universidad de Buenos Aires.

19 de julio de 2010

El Orador

- Fernando Benavides

El camino recorrido

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

La idea

- Desarrollar una herramienta para los programadores funcionales como las que existen en el paradigma de orientación a objetos

El Orador

- Fernando Benavides

El camino recorrido

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

La idea

- Desarrollar una herramienta para los programadores funcionales como las que existen en el paradigma de orientación a objetos

El Orador

- ▶ Fernando Benavides

El camino recorrido

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

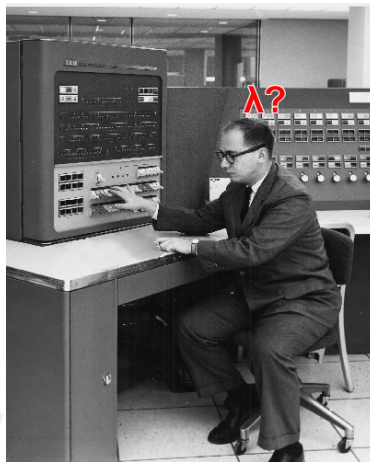
La idea

- ▶ Desarrollar una herramienta para los programadores funcionales como las que existen en el paradigma de orientación a objetos

Trabajando en *Haskell*

¿Cómo trabaja un desarrollador *Haskell*?

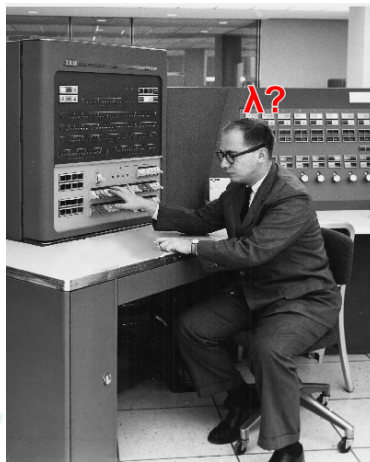
- ▶ Crea o modifica módulos con su editor de texto favorito
- ▶ Los compila utilizando *GHC*
- ▶ Genera paquetes con *Cabal*
- ▶ Para realizar pruebas, recurre a *GHCi*



Trabajando en *Haskell*

¿Cómo trabaja un desarrollador *Haskell*?

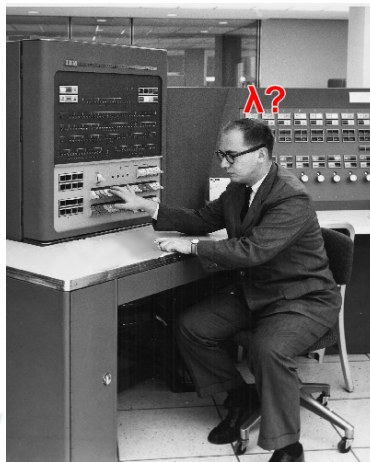
- ▶ Crea o modifica módulos con su editor de texto favorito
- ▶ Los compila utilizando *GHC*
- ▶ Genera paquetes con *Cabal*
- ▶ Para realizar pruebas, recurre a *GHCi*



Trabajando en *Haskell*

¿Cómo trabaja un desarrollador *Haskell*?

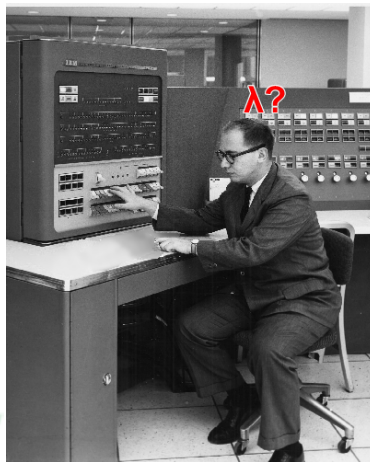
- ▶ Crea o modifica módulos con su editor de texto favorito
- ▶ Los compila utilizando *GHC*
- ▶ Genera paquetes con *Cabal*
- ▶ Para realizar pruebas, recurre a *GHCi*



Trabajando en *Haskell*

¿Cómo trabaja un desarrollador *Haskell*?

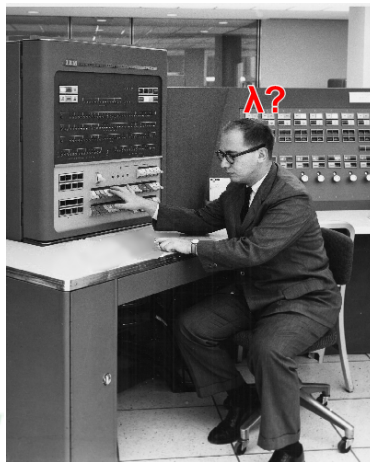
- ▶ Crea o modifica módulos con su editor de texto favorito
- ▶ Los compila utilizando *GHC*
- ▶ Genera paquetes con *Cabal*
- ▶ Para realizar pruebas, recurre a *GHCi*



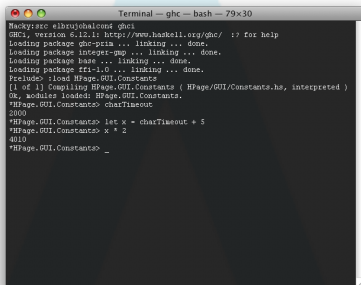
Trabajando en *Haskell*

¿Cómo trabaja un desarrollador *Haskell*?

- ▶ Crea o modifica módulos con su editor de texto favorito
- ▶ Los compila utilizando *GHC*
- ▶ Genera paquetes con *Cabal*
- ▶ Para realizar pruebas, recurre a *GHCi*



GHCi



```
MacOS:~$ ghc -i. ghc
GHCi, version 6.12.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-paths ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
Loading package ffi-1.0 ... linking ... done.
Prelude> load HPage.GUI.Constants
[1 of 1] Compiling HPage.GUI.Constants ( HPage/GUI/Constants.hs, interpreted )
OK, modules loaded: HPage.GUI.Constants.
*HPage.GUI.Constants> charTimeout
2000
*HPage.GUI.Constants> let x = charTimeout + 5
*HPage.GUI.Constants> x * 2
4010
*HPage.GUI.Constants> _
```

GHCi permite:

- ▶ introducir código para ejecutarlo y observar los resultados obtenidos
- ▶ definir expresiones y utilizarlas
- ▶ cargar módulos para utilizar sus funciones, tipos de datos, etc.

Trabajando con Lenguajes Orientados a Objetos



En cambio quienes programan en *Java*, *.NET* o *Smalltalk* cuentan con una *IDE* que provee

- ▶ Autocompleción de código
- ▶ Compilación automática
- ▶ Debugger integrado
- ▶ Herramientas para "micro-testing"

Trabajando con Lenguajes Orientados a Objetos



En cambio quienes programan en *Java*, *.NET* o *Smalltalk* cuentan con una *IDE* que provee

- ▶ Autocompleción de código
- ▶ Compilación automática
- ▶ Debugger integrado
- ▶ Herramientas para *"micro-testing"*

Trabajando con Lenguajes Orientados a Objetos



En cambio quienes programan en *Java*, *.NET* o *Smalltalk* cuentan con una *IDE* que provee

- ▶ Autocompleción de código
- ▶ Compilación automática
- ▶ Debugger integrado
- ▶ Herramientas para *"micro-testing"*

Trabajando con Lenguajes Orientados a Objetos



En cambio quienes programan en *Java*, *.NET* o *Smalltalk* cuentan con una *IDE* que provee

- ▶ Autocompleción de código
- ▶ Compilación automática
- ▶ Debugger integrado
- ▶ Herramientas para *"micro-testing"*

Trabajando con Lenguajes Orientados a Objetos



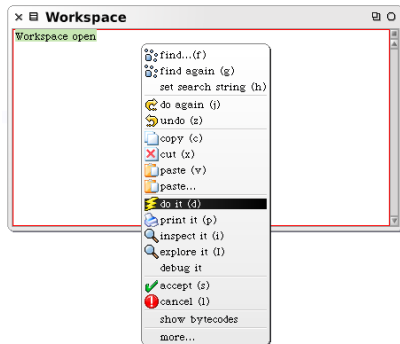
En cambio quienes programan en *Java*, *.NET* o *Smalltalk* cuentan con una *IDE* que provee

- ▶ Autocompleción de código
- ▶ Compilación automática
- ▶ Debugger integrado
- ▶ Herramientas para “*micro-testing*”

“Micro-testing”

El *Workspace* de *Smalltalk* permite:

- ▶ introducir código para ejecutarlo, inspeccionarlo y analizar los resultados obtenidos
- ▶ administrar varias paginas de texto
- ▶ crear objetos y utilizarlos



Conociendo λ Page

Como el Workspace de Smalltalk ...

λ Page es similar al *Workspace* de *Smalltalk* pues permite al usuario

- ▶ Evaluar expresiones
- ▶ Detectar excepciones
- ▶ Administrar páginas de texto libre
- ▶ Intercalar expresiones y definiciones

... pero para Haskell

Pero, a su vez, por estar hecho para *Haskell*, presenta otros desafíos

- ▶ *Lazy evaluation*
- ▶ Expresiones puras vs. Expresiones con efectos
- ▶ Administración de módulos

... pero para Haskell

Pero, a su vez, por estar hecho para *Haskell*, presenta otros desafíos

- ▶ *Lazy evaluation*
- ▶ Expresiones puras vs. Expresiones con efectos
- ▶ Administración de módulos

... pero para Haskell

Pero, a su vez, por estar hecho para *Haskell*, presenta otros desafíos

- ▶ *Lazy evaluation*
- ▶ Expresiones puras vs. Expresiones con efectos
- ▶ Administración de módulos

λ Page *por Dentro*

Desarrollo de $\lambda Page$

- ▶ $\lambda Page$ está desarrollado en *Haskell*
- ▶ En gran parte está desarrollado utilizando $\lambda Page$
- ▶ Se conecta con *GHC* a través de su API
- ▶ Su interfaz gráfica fue creada usando *wxHaskell*
- ▶ Su alto grado de paralelismo se logra utilizando *eprocess*

Desarrollo de λ Page

- ▶ λ Page está desarrollado en *Haskell*
- ▶ En gran parte está desarrollado utilizando λ Page
- ▶ Se conecta con *GHC* a través de su API
- ▶ Su interfaz gráfica fue creada usando *wxHaskell*
- ▶ Su alto grado de paralelismo se logra utilizando *eprocess*

Desarrollo de λ Page

- ▶ λ Page está desarrollado en *Haskell*
- ▶ En gran parte está desarrollado utilizando λ Page
- ▶ Se conecta con *GHC* a través de su API
- ▶ Su interfaz gráfica fue creada usando *wxHaskell*
- ▶ Su alto grado de paralelismo se logra utilizando *eprocess*

Desarrollo de λ Page

- ▶ λ Page está desarrollado en *Haskell*
- ▶ En gran parte está desarrollado utilizando λ Page
- ▶ Se conecta con *GHC* a través de su API
- ▶ Su interfaz gráfica fue creada usando *wxHaskell*
- ▶ Su alto grado de paralelismo se logra utilizando *eprocess*

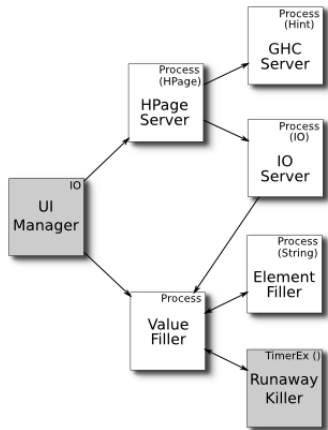
Desarrollo de λ Page

- ▶ λ Page está desarrollado en *Haskell*
- ▶ En gran parte está desarrollado utilizando λ Page
- ▶ Se conecta con *GHC* a través de su API
- ▶ Su interfaz gráfica fue creada usando *wxHaskell*
- ▶ Su alto grado de paralelismo se logra utilizando *eprocess*

Arquitectura

Principales Requerimientos:

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



Ejemplo de Interacción

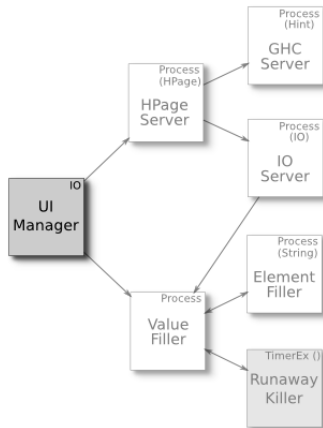
Veremos cómo interactúan estos componentes para evaluar la siguiente expresión:

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

Ejemplo de Interacción

Procesos Involucrados:

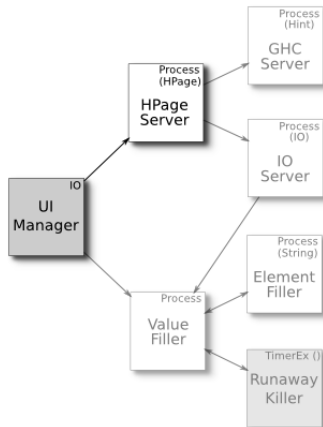
- UI Manager **operando**



Ejemplo de Interacción

Procesos Involucrados:

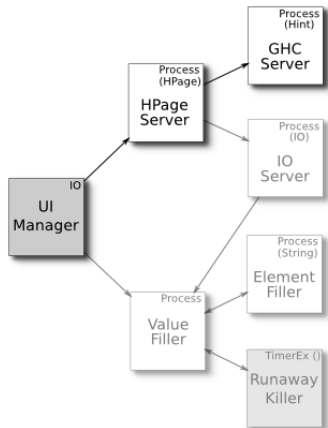
- ▶ UI Manager esperando
- ▶ HPage Server **operando**



Ejemplo de Interacción

Procesos Involucrados:

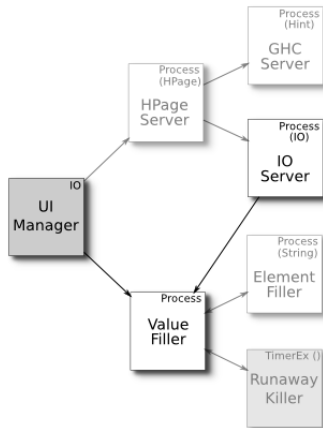
- ▶ UI Manager esperando
- ▶ HPage Server esperando
- ▶ GHC Server **operando**



Ejemplo de Interacción

Procesos Involucrados:

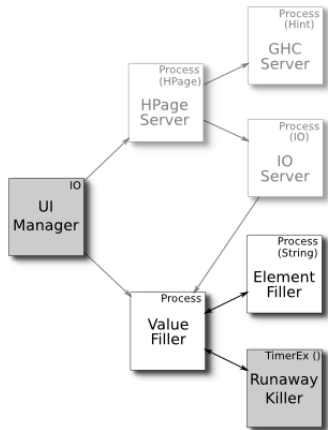
- ▶ UI Manager **operando**
- ▶ IO Server **operando**
- ▶ Value Filler esperando



Ejemplo de Interacción

Procesos Involucrados:

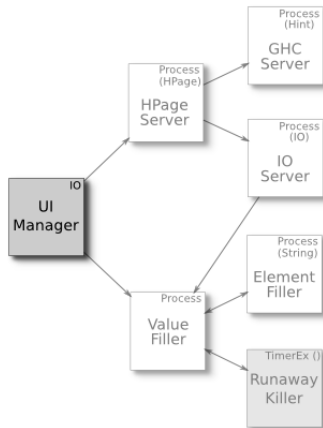
- ▶ UI Manager **operando**
- ▶ Value Filler esperando
- ▶ Element Filler **operando**
- ▶ Runaway Killer **operando**



Ejemplo de Interacción

Procesos Involucrados:

- UI Manager **operando**



Próximos Pasos

Limitaciones

- ▶ Más tipos *especiales*
 - ▶ Tuplas
 - ▶ Either
 - ▶ Maybe
- ▶ Composición
 - ▶ Listas de listas
 - ▶ Acciones que generen listas
 - ▶ Listas de acciones
- ▶ Nuevas visualizaciones
 - ▶ Más que un cuadro de texto

¿Qué se puede hacer?

- ▶ Clase Presentable

Limitaciones

- ▶ Más tipos *especiales*
 - ▶ Tuplas
 - ▶ Either
 - ▶ Maybe
- ▶ Composición
 - ▶ Listas de listas
 - ▶ Acciones que generen listas
 - ▶ Listas de acciones
- ▶ Nuevas visualizaciones
 - ▶ Más que un cuadro de texto

¿Qué se puede hacer?

- ▶ Clase Presentable

Limitaciones

- ▶ Más tipos *especiales*
 - ▶ Tuplas
 - ▶ Either
 - ▶ Maybe
- ▶ Composición
 - ▶ Listas de listas
 - ▶ Acciones que generen listas
 - ▶ Listas de acciones
- ▶ Nuevas visualizaciones
 - ▶ Más que un cuadro de texto

¿Qué se puede hacer?

- ▶ Clase Presentable

Limitaciones

- ▶ Más tipos *especiales*
 - ▶ Tuplas
 - ▶ Either
 - ▶ Maybe
- ▶ Composición
 - ▶ Listas de listas
 - ▶ Acciones que generen listas
 - ▶ Listas de acciones
- ▶ Nuevas visualizaciones
 - ▶ Más que un cuadro de texto

¿Qué se puede hacer?

- ▶ Clase Presentable

Otras Herramientas

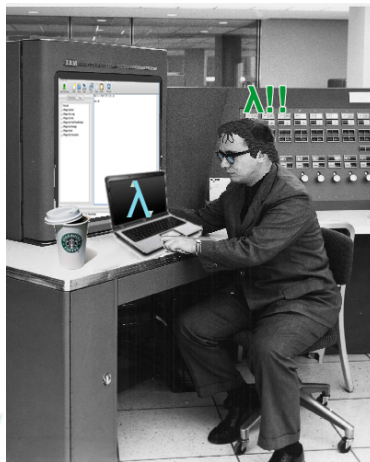
Con λ Page hemos acercado al desarrollador *Haskell* sólo **una** de muchas herramientas:

- ▶ Mejores herramientas para TDD
- ▶ Refactoring
- ▶ Análisis de Terminación
- ▶ ...

Otras Herramientas

Con λ Page hemos acercado al desarrollador *Haskell* sólo **una** de muchas herramientas:

- ▶ Mejores herramientas para TDD
- ▶ Refactoring
- ▶ Análisis de Terminación
- ▶ ...



Agradecimientos / Preguntas

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