λ*Page*

Fernando Benavides

Departamento de Computación, FCEyN,Universidad de Buenos Aires.

13 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

λPage

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*

Contexto

Presenciamos actualmente la aparición de aplicaciones desarrolladas dentro del *paradigma funcional*:

- CouchDB
- ejabberd
- Chat de Facebook

Desarrolladas en lenguajes como:

- Erlang
- Haskell

Contexto

Presenciamos actualmente la aparición de aplicaciones desarrolladas dentro del *paradigma funcional*:

- CouchDB
- ejabberd
- Chat de Facebook

Desarrolladas en lenguajes como:

- Erlang
- Haskell

Necesidades

Para desarrollar en *Haskell* existen herramientas como:

- GHCi
- Hugs
- Hat

Pero no existen herramientas como

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

Necesidades

Para desarrollar en *Haskell* existen herramientas como:

- GHCi
- Hugs
- Hat

Pero no existen herramientas como

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

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]::[Float]
```

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

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

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

Interpretación de Expresiones

λPage permite interpretar expresiones como:

```
[1,2,3]::[Float]
```

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

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

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

Interpretación de Expresiones

λPage permite interpretar expresiones como:

```
[1,2,3]::[Float]
```

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

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

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

Interpretación de Expresiones

λPage permite interpretar expresiones como:

```
[1,2,3]::[Float]
```

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

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

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

Presentación de Resultados

- Para expresiones inválidas, *λPage* presenta el error informado por *GHC*
- Para expresiones sin resultado "visible", *λPage* permite conocer su tipo
- Para expresiones infinitas, *λPage* presenta su valor incrementalmente hasta que el usuario cancela la evaluación
- Para expresiones que requieren cálculos infinitos, *λPage* permite al usuario cancelar la evaluación

Acciones con Efectos Colaterales

En *Haskell* para realizar acciones que puedan generar efectos colaterales 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.
λPage toma el modelo de *GHCi* y ejecuta la acción, presentando su resultado

Acciones con Efectos Colaterales

En *Haskell* para realizar acciones que puedan generar efectos colaterales 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.

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

Acciones con Efectos Colaterales

En *Haskell* para realizar acciones que puedan generar efectos colaterales 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.
λ*Page* 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]`)
Por ejemplo, al evaluar la siguiente expresión:

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

λPage podría presentar como resultado

[1,

o, en caso de detectar excepciones en los elementos de la lista,

[1, ⊥, 2 ⊥, 3,

Listas

λPage interpreta de modo particular las listas (expresiones de tipo `Show a => [a]`)
Por ejemplo, al evaluar la siguiente expresión:

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

λPage podría presentar como resultado

[1,

o, en caso de detectar excepciones en los elementos de la lista,

[1, ⊥, 2 ⊥, 3,

Listas

λPage interpreta de modo particular las listas (expresiones de tipo `Show a => [a]`)
Por ejemplo, al evaluar la siguiente expresión:

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

λPage podría presentar como resultado

[1,

o, en caso de detectar excepciones en los elementos de la lista,

[1, ⊥, 2, ⊥, 3,

Listas

λPage interpreta de modo particular las listas (expresiones de tipo `Show a => [a]`)
Por ejemplo, al evaluar la siguiente expresión:

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

λPage presenta como resultado:

$$[1, \bot, 2, \bot, 3, \bot, 4]$$

Otras Características

TODO: Paralelismo
TODO: Hablar de la integración con Cabal y Hayoo!
TODO: Hablar de Importar/Cargar/Recargar Módulos
TODO: Hablar de Género

Arquitectura

TODO: Gráfico de la arquitectura y su explicación

Secuencia

TODO: Gráfico de la secuencia y su explicación?

Principales Decisiones de Diseño

TODO: Principales temas de diseño

Implementación

TODO: Principales temas de implementación

Objetivos Alcanzados

TODO: Objetivos Alcanzados

Logros Adicionales

TODO: Otras cosas logradas más allá de lo que originalmente nos propusimos

Tareas a Realizar

TODO: Trabajo a Futuro

¡Gracias a todos!

¿En qué los puedo ayudar?