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 varias aplicaciones desarrolladas dentro del *paradigma funcional*:

- CouchDB
- ejabberd
- Chat de Facebook

Desarrolladas en lenguajes como:

- Erlang
- Haskell

Contexto

Presenciamos actualmente la aparición de varias 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]
```

```
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]
```

```
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]
```

```
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]
```

```
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

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 λ*Page* 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]
```

λ*Page*, 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

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*

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*

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*

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?