Tesis de Licenciatura

λPage

Un bloc de notas para desarrolladores Haskell

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Alumno

Fernando Benavides (LU 470/01)

greenmellon@gmail.com

Directores

Dr. Diego Garbervetsky

Lic. Daniel Gorín

Abstract

El presente documento describe una herramienta para desarrolladores *Haskell* que pretende facilitar la tarea de “debuggear”, analizar y entender código, llamada λ***Page***. Con ella el usuario puede manipular “páginas” de texto libre que contengan expresiones *Haskell* , intentar interpretar éstas expresiones independientemente y analizar los resultados obtenidos.

Índice

1. Introducción

1.1. Motivación

Actualmente estamos presenciando un importante cambio en el desarrollo de sistemas, gracias al éxito de proyectos como CouchDB[1], ejabberd[2] y el chat de Facebook[3], todos ellos desarrollados utilizando lenguajes del paradigma funcional.

Ejemplos de éstos lenguajes de programación, como Haskell[4] o Erlang[5], demuestran ser maduros, confiables y presentan claras ventajas en comparación con los lenguajes tradicionales del paradigma imperativo. Sin embargo, los desarrolladores que deciden realizar el cambio de paradigma se encuentran con el problema de la escasez de ciertas herramientas que les permitan realizar su trabajo más eficientemente. Por el contrario, éstas herramientas abundan en el desarrollo de proyectos utilizando lenguajes orientados a objetos. En particular, nuestro foco de atención se centra sobre aquellas herramientas que permiten realizar *debugging* y *entendimiento* de código a través de *"micro-testing"*[6] .

Los desarrolladores Haskell cuentan actualmente con dos herramientas de este tipo:

GHCi[7] La consola que provee GHC[8] permite a los desarrolladores evaluar expresiones, verificar su tipo o su clase. Cuenta también con un mecanismo de debugging[9] integrado que permite realizar la evaluación de expresiones paso a paso. Pese a ser la herramienta más utilizada por los desarrolladores, *GHCi* tiene varias limitaciones. En particular:

- No permite editar más de una expresión a la vez
- No permite intercalar expresiones con definiciones
- Si bien permite utilizar definiciones, éstas se pierden al recargar módulos
- No es sencillo utilizar en una sesión las definiciones y/o expresiones creadas en sesiones anteriores

Hat[10] Un herramienta para realizar seguimiento a nivel de código fuente. A través de la generación de trazas de ejecución, *Hat* ayuda a localizar errores en los programas y es útil para entender su funcionamiento. Sin embargo, por estar basado en la generación de trazas, requiere la compilación y ejecución de un programa para poder utilizarlo y esto no siempre es cómodo para el desarrollador que puede querer simplemente analizar una expresión particular que incluso quizá no compile aún. Además, su mantenimiento activo parece haber cesado hace más de un año y en su página se observa una importante lista de problemas conocidos[11] y características deseadas[12].

---

[1] http://couchdb.apache.org

[2] http://www.ejabberd.im

[3] http://www.facebook.com

[4] http://www.haskell.org

[5] http://www.erlang.org

[6] Entiéndase "micro-testing" como la tarea de realizar tests eventuales para entender o evaluar algún aspecto de un programa

[8] http://www.haskell.org/ghc

[9] http://www.haskell.org/ghc/docs/6.10-latest/html/users_guide/ghci-debugger.html

[11] http://www.haskell.org/hat/bugs.html

[12] http://www.haskell.org/hat/bugs.html

1.2. Trabajos Relacionados

En el mundo de la programación orientada a objetos podemos encontrar herramientas de este tipo, como Java Scrapbook Pages[13] para Java[14] y Workspace[15] para SmallTalk[16]. Utilizando estos aplicativos, los desarrolladores pueden introducir pequeñas porciones de código, ejecutarlas y luego inspeccionar y analizar los resultados obtenidos. Un concepto compartido por ambas herramientas es el de presentar “páginas” de texto en las que varias expresiones pueden intercalarse con partes de texto libre y permitir al desarrollador intentar evaluar sólo una porción de todo lo escrito. Estas páginas pueden ser guardardas y luego recuperadas de modo de poder analizar nuevamente las mismas expresiones. Además permiten crear objetos (lo que para los lenguajes funcionales equivaldría a definir expresiones) locales a la página en uso y utilizarlos en ella.

Dentro del paradigma funcional, con un enfoque similar, aunque un poco más orientado a la presentación y visualización de documentos, Keith Hanna[17] de la Universidad de Kent, ha desarrollado Vital[18]. *Vital* es una implementación de un entorno de visualización de documentos para *Haskell* . Pretende presentar *Haskell* de una manera apropiada para usuarios finales en areas de aplicación como la ingeniería, las matemáticas o las finanzas. Dentro de esta herramienta, los módulos *Haskell* son presentados como documentos en los que pueden visualizarse los valores que en ellos se definen directamente en el lugar en el que aparecen, ya sea de modo textual o gráfico (como “vistas”).

1.3. λ*Page*

---

[13] http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.jdt.doc.user/reference/ref-34.htm
[14] http://www.java.com
[15] http://wiki.squeak.org/squeak/1934
[16] http://www.smalltalk.org
[17] http://www.cs.kent.ac.uk/people/staff/fkh/
[18] http://www.cs.kent.ac.uk/projects/vital/

2. Tutorial - Descubriendo λ***Page***

2.1. Instalación

TODO: Instrucciones generales y el resto copiar (y verificar) de la wiki

2.1.1. OSX

2.1.2. Windows

2.1.3. Linux

2.2. QuickStart

TODO: Tutorial donde se noten las features de λ***Page***

3. Desarrollo - ¿Cómo se hizo λ*Page*?

3.1. Arquitectura General

TODO: Gráficos de arquitectura general

3.2. Diseño

TODO: Contar las decisiones que tomamos y por qué

3.3. Implementación

TODO: Detalles generales de implementación

3.3.1. wxHaskell

TODO: Pros y contras y workarounds

3.3.2. Bottoms

TODO: Cómo manejamos los bottoms en el resultado?

3.3.3. Threads

TODO: Cómo manejamos los threads para la GUI y la VM?

3.3.4. hint

TODO: Cómo utilizamos hint para conectarnos con la VM y el tema de que es *lazy*

3.4. Problemas Resueltos

3.4.1. Un Editor de Texto en Haskell

TODO: wxhNotepad

3.4.2. Multithreading en GHC

TODO: ¿Cómo simular multhreading cuando GHC no es multithread? TODO: Otros

4. Resultados

4.1. Objetivos Alcanzados

TODO: ¿Qué se puede hacer ahora que existe λ***Page***?

4.2. Trabajo a Realizar

TODO: Future Work