FACULTAD DE CIENCIAS EXACTAS, INGENIERIA Y AGRIMENSURA

Licenciatura en Ciencias de la Computación
Análisis de Lenguajes de Programación
Prof Dr. Mauro Jaskelioff

>>> Resumidor de Noticias

Matías Saper

19 de junio de 2017

Av. Pellegrini 250
Rosario
Santa Fe
Argentina

Tél. (+54) 0341- 4802649

www.fceia.unr.edu.ar

Índice general

Resumidor de noticias 1
Indice 2
1. Introducción 3
2. Motivación 3
3. Módulo Main 3
3.1. Funciones Principales 3
4. Módulo Scraper 4
4.1. Funciones Principales 4
5. Módulo TinyParser 5
5.1. Funciones Principales 5
6. Módulo Data 5
6.1. Datas 5
7. Módulo Configure 6
7.1. Funciones Principales 6
8. Conclusión 7

1. Introducción

En este informe se explica la función de cada uno de los módulos que forman parte del programa, describiendo cada uno de sus componentes, analizando las decisiones de diseño, librerías utilizadas y funciones principales. En el código fuente además, se puede encontrar breves descripciones de cada una de las funciones. El proyecto también cuenta con un manual de uso adjunto con el programa.

2. Motivación

La idea principal del trabajo es poder tener una visión de los principales titulares de los portales de noticias de distintos diarios que cuenten con servicio RSS, sin necesidad de tener que ingresar a la página de éstos, evitando cargar todo el contenido, el cual incluye, en general, demasiada publicidad. Este programa nos permite configurar las prioridades de las noticias a mostrar, es decir, tenerlas en distintos grupos para poder verlas de una forma más ordenada, agregar o quitar páginas, abrir algún título que deseemos ampliar para ver el contenido total (utiliza firefox) y nos provee opciones gráficas como cambiar el color de la fuente de texto y color de fondo. Todos estos datos se guardan en archivos de configuración y luego se cargan al ingresar nuevamente al programa.

3. Módulo Main

Este módulo se encarga de la interacción con el usuario, es decir, de la entrada/salida. Además, invoca las funciones necesarias para configurar nuestro programa de la manera deseada.

Al iniciarse el programa, este módulo limpia la pantalla, carga nuestra configuración guardada (en caso de que exista, de lo contrario crea una por defecto), carga el archivo de noticias y finalmente muestra el menú interactivo en el cual el usuario puede tomar sus propias decisiones sobre cómo seguir con el programa.

3.1. Funciones Principales

- *main :: IO ()*

  Esta función es la encargada de llamar a las demás funciones del módulo las cuales cargan las configuraciones, llama al menú interactivo y despliega el titulo del programa adaptable al tamaño de la terminal.

- *menu :: ([Config],Prior) ->IO ()*

  Esta función maneja el menú principal del programa. Desde este podemos ir al menú de noticias, menú de RSS, Cambiar las opciones gráficas, restaurar valores predeterminados y salir del programa.

- *agregarLinks :: ([Config],Prior) ->IO ()*

  Toma como parámetro una tupla con las configuraciones y la lista de urls de cada prioridad (Prior), le pedirá al usuario ingresar una URL y una prioridad y finalmente nos agrega la URL ingresada a la lista de prioridades a través de la función agregarUrlConf.

- *verNoticias :: News ->IO ()*

  Función que muestra en pantalla las noticias descargadas anteriormente.[1] Tener en cuenta que

[1] La decisión de mostrar y descargar las noticias de forma separada tuvo como motivo el hecho de que no podemos intuir cuando habrá una noticia nueva, por lo que le deja al usuario la libertad de elegir cuando actualizar las noticias. Otro factor es que si tenemos un conjunto de páginas grande, y lo actualizamos en un periodo de tiempo constante, podría no llegar a ver todos los títulos y actualizarse el programa, lo cual dificulta la visión de la lista de títulos. Además tendríamos que descargar nuevamente todos los titulares, lo que puede llevar un tiempo considerable.

esto implica que si realizamos algún cambio en los conjuntos de prioridades, debemos actualizar para ver los cambios.

- *actNoticias :: Prior->News ->IO ()*

  Actualiza las noticias de la prioridad elegida, descargando la información del conjunto de páginas correspondiente y actualiza el archivo de noticias.

- *irUrl :: Priority ->News ->IO ()*

  Esta función espera la elección del usuario de un número de la lista de noticias y en caso de ser correcto, ejecuta un *runCommand "firefox url"*, donde la url es la obtenida en base al índice elegido por el usuario. Utiliza la función *parsercito* para corroborar la correctidud de la entrada hecha por el usuario. Además tiene un manejo de errores en caso de ingresar un índice inválido.

- *eliminarRss :: Prior ->[Config] ->IO ()*

  Elimina un Link RSS ingresado manualmente por el usuario. Si el link forma parte del programa, en caso contrario, nos avisará con un error que el link es incorrecto.

- *graphOptions :: Prior ->IO ()*

  Esta función despliega un menú para que el usuario pueda elegir sobre los aspectos gráficos del programa.

4. Módulo Scraper

Este módulo es el encargado de descargar la información del enlace RSS indicado, utilizando la librería Feed y HTTP.

4.1. Funciones Principales

- *getResponseRss :: String ->IO String*

  Se encarga de obtener la información decodificada. Primeramente se aplica getRequest, que efectúa una request de tipo get, el resultado es tomado por simpleHTTP, que trasmite y recoge la response como *Result*. Luego getResponseBody obtiene el cuerpo de esta respuesta, a la que finalmente decodificamos con decodeString perteneciente a la librería *Codec.Binary.UTF8.String*

- *getTuples :: String ->IO (Maybe [(Maybe String, Maybe String)])*

  Esta funcion se encarga de obtener las tuplas con la información del titulo y la Url. Utiliza el operador <$>. Se comienza obteniendo la información decodificada vía *getResponseRss*, luego se parsea el contenido y se transforma en estructura Feed (esto aprovecha el hecho de que sabemos como se compone un RSS, con lo que se crea una estructura con los elementos que forman parte del RSS, para después filtrarla con los elementos que nos interese, en este caso los títulos y su link asociado) para poder manejarla a través de *fmap feedItems* que nos devuelve algo del tipo *IO [(Maybe String,Maybe String)]* y finalmente con *fmap (map getTitleAndUrl)* obtenemos lo que queremos, una tupla con el titulo de nuestra noticia y el link correspondiente.

- *extractData :: Maybe [(Maybe String, Maybe String)] ->[(String,String)]*

  Esta función simplemente nos "limpia"los Maybe de la estructura o nos muestra un error en caso de encontrar un Nothing el cual podría ser producto de un error de parseo o decodificación en algún momento de la obtención y depuración de los datos.

- *scrap :: String ->IO [(String,String)]*

  Finalmente esta función nos devuelve una lista de tuplas con el titulo y su url.

5. Módulo TinyParser

Este modulo es el encargado del parseo de los archivos generados para guardar la información sobre nuestro programa. Parsea el archivo de configuración y el archivo de noticias. Utiliza el módulo Parsing provisto en clases con un pequeño agregado de space',char' .[2]

5.1. Funciones Principales

- *p1 :: [Config] ->Prior ->Parser ([Config],Prior)*

  La funcion p1 es la encargada de realizar el parser general para el archivo config. Utiliza parsers mas chicos como fond y font que parsean las String representantes del fondo y la fuente (colores de la terminal), urlP que parsea los datas Prior y parsers de URLs. Con este parser obtenemos la tupla con las configuraciones y la estructura de prioridades.

- *parseNews :: News ->Parser News*

  parseNews tiene una utilidad similar a p1, parsea el archivo de noticias utilizando parser mas chicos internamente como altaNew, mediaNew y bajaNew que analizan las news de distintas prioridades. Además de utilizar otros parsers auxiliares para poder aceptar las tuplas. AuxN es la que nos permite que distintos caracteres puedan formar parte de las noticias .

6. Módulo Data

Es el encargado de la definición de los datos utilizados en el programa. Ademas posee algunas funciones para manejar estas estructuras.

6.1. Datas

- ```
  data Prior = P {  a :: [Url],
                    m :: [Url],
                    b :: [Url] } deriving Show
  ```

  Prior es un record utilizado para guardar las Urls RSS que el usuario agregue. Las letras a,m,b representan los links de prioridad alta, media y baja respectivamente. Este tipo fue definido como record para poder acceder facilmente al contenido sin tener que definir por separado sus funciones, las cuales devolven la lista de Urls correspondiente a su prioridad.

- ```
  data Priority = Alta | Media | Baja deriving Show
  ```

  Priority representa esencialmente lo mismo que Prior. La idea de este dato es poder utilizar y llamar funciones que se comporten de forma distinta para distintas Urls, dependiendo de su prioridad, sin necesidad de pasar información alguna sobre éstas.

- ```
  data Config = Fondo Int Int | Fuente Int Int deriving Show
  ```

  Config es un tipo de datos creado para guardar los valores de seleccionados como configuracion grafica. *Fondo Int Int* indica que el fondo de la terminal tiene un color y una opacidad definida por el usuario, mientras que *Fuente Int Int* representa el color y opacidad de la las letras de la terminal.

- ```
  data News = N {  na :: ([(String,Url)],Int),
                   nm :: ([(String,Url)],Int),
                   nb :: ([(String,Url)],Int) } deriving Show
  ```

[2] En el código fuente esta la gramática a grosso modo de los parser utilizados.

El tipo de dato *News* fue creado para tener un fácil acceso a los títulos noticias, su respectivo link y la cantidad total de noticias de esa prioridad obtenidas. Las noticias de prioridad alta (na), las noticias de prioridad media (nm), las noticias de prioridad baja (nb). Los títulos son Strings que fueron parseados luego de ser descargados por el scraper con las funciones anteriormente descritas y luego se guardaran en este dato, el cual se recupera cuando sea necesario mostrarlo.

- *addUrl :: Url ->Priority ->Prior ->IO Prior*

  Función que agrega una Url a la lista, en su correspondiente prioridad y asegurándose que no exista previamente.

- *showUrls :: Prior ->IO ()*

  Muestra por pantalla las Urls de la prioridad pasada como argumento.

- *checkUrl :: Url ->IO ()*

  Checkea el estado de una página y nos muestra por pantalla el resultado, hay que tener en cuenta que el se utiliza *simpleHTTP* el cual no funciona con HTTPS por lo que por mas que se encuentre ONLINE, nos indicara que esta OFFLINE.

7. Módulo Configure

Este es el módulo más importante del programa ya que se encarga de unir las funcionalidades de todos los módulos, dejar los datos de forma correcta para que el Main luego puede trabajar con los datos correctamente.

Primeramente nos encontraremos que el módulo posee distintas constantes inicializadas, esto nos servirá para ubicar los archivos de configuración y restaurar valores por default.

7.1. Funciones Principales

- *evalConf :: Config ->IO ()*

  Evalua la configuración pasada como argumento, utiliza *setSGR* para cambiar los colores en la terminal.

- *findNews :: IO News*

  Parsea las noticias desde el archivo de noticias, asegurándose de su existencia, en caso de que no exista crea el archivo un archivo con contenido default.

- *restoreDefault :: IO ()*

  Utiliza funciones que se encargan de restaurar los archivos como por default y luego procesa el archivo grafico con *procesarConf*.

- *procesarConf :: IO ([Config],Prior)*

  Lee el archivo cfg (configuracion gráfica y prioridades de las Urls), luego parsea el contenido con las funciones de parseo explicadas anteriormente (a través de *p1*) y finalmente evalúa el resultado con *evalConf*.

- *changeConfigCol :: Prior ->[Int] ->IO ()*

  Cambia el archivo de configuración, utiliza un archivo temporal para evitar problemas de LOCK. Escribe como String en un formato que el procesador de configuraciones(*procesarConf*) puede comprender.

- *agregarUrlConf :: Url ->Priority ->([Config],Prior) ->IO ()*

  Agrega una Url al archivo de Config, al igual que la función anterior, evita problemas de apertura y escritura de archivos, escribe en el formato correcto y finalmente procesa la información.

- *showNews :: Priority ->IO Int*

  ShowNews toma una *Priority* y busca todas las noticias que hayan sido descargadas en el archivo notis, en caso de no encontrar noticias de la prioridad requerida, nos muestra un cartel que nos lo indica, si encuentra noticias, se imprime con una funcion auxiliar que nos muestra el numero que la noticia ocupa para luego poder ampliarla en el navegador.

- *updateNews :: Priority ->Prior ->News ->IO Int*

  Esta función se encarga, a partir de una Prioridad y una lista, de actualizar las noticias en el archivo de noticias. Primeramente filtra las noticias requeridas, luego corrobora que exista algún diario de esa prioridad, si existe, llama a una función auxiliar *auxParse :: [Url] ->IO [(String, Url)]* que se encargara del scraping llamando a la función *scrap* del módulo scraper. Esta llamada nos devolverá una lista de tuplas con la noticia y su correspondiente Url. Finalmente se llama a *writeNews* que actualiza el archivo de noticias.

- *writeNews :: Priority ->[(String,Url)] ->Prior ->News ->IO ()*

  Escribe en el archivo de configuración las nuevas noticias, al igual que *changeConfigCol* y *agregarUrlConf* con la sintaxis requerida para que los parser pueda entenderlo y en un archivo temporal para luego transformarlo en el final para evitar los locks.

8. Conclusión

En este trabajo se intento aplicar los temas aprendidos en la materia de *Análisis del lenguaje de Programación* tales como los parsers. Se utilizó el conocimiento sobre mónadas aprovechando la Monada IO y sus operadores que facilitaron la creación de funciones y formas de mostrar la información al usuario, además de la toma decisiones sobre como representar datos. La idea del trabajo fue realizar un EDSL, que puede verse en los archivos de configuraciones (Noticias.cfg y Config.cfg) el cual se puede modificar y a través de parsers, funciones evaluación y procesado, obtener un resultado deseado.

Por otro lado quizás hubiese sido más adecuado a la consigna de EDSL agregar comandos como por ejempllo #ADD y #REMOVE para los links de las URLs y poder utilizarlos dentro de los archivos de configuraciones, pero decidí hacerlo gráfico mediante la consola para que el usuario pueda hacerlo de manera interactiva. Igualmente se podría agregar este tipo de comandos añadiendo mas funciones de parseo, ya que la funcionalidad esta dada internamente, solo que en vez de ejecutarse como resultado de un parseo, se llama al momento de que el usuario elija la opción agregar o remover de forma manual en el menú.

Bibliografía

[1] Hackage de Feed. https://hackage.haskell.org/package/feed

[2] Hackage de HTTP. http://hackage.haskell.org/package/HTTP-3000.0.0

[3] Hackage de Console-Terminal-Size. https://hackage.haskell.org/package/terminal-size-0.3.2.1/docs/System-Console-Terminal-Size.html

[4] Hackage de Codec Binary UTF8 String. https://hackage.haskell.org/package/utf8-string-1.0.1.1/docs/Codec-Binary-UTF8-String.html

[5] StackOverflow. Pregunta sobre Rss downloader con código. http://stackoverflow.com/questions/17038947/simple-rss-downloader-in-haskell/17044041#17044041

[6] Librería de Parser. http://comunidades.campusvirtualunr.edu.ar/pluginfile.php/219952/mod_resource/content/2/Parsing.lhs

[7] Hackage de directory. https://hackage.haskell.org/package/directory-1.3.1.1/docs/System-Directory.html

[8] Hackage de network-Uri. https://hackage.haskell.org/package/network-uri

[9] Hackage de unicode-show. https://hackage.haskell.org/package/unicode-show

[10] Información sobre HTML Parsing con cursores. https://github.com/snoyberg/xml/issues/70

[11] Información sobre HTML Parsing con cursores 2. https://www.schoolofhaskell.com/school/starting-with-haskell/libraries-and-frameworks/text-manipulation/tagsoup

[12] Información sobre Monada IO. http://book.realworldhaskell.org/read/io.html

[13] Información sobre archivos Cabal. https://www.haskell.org/cabal/users-guide/developing-packages.html

[14] LaTeX web para el informe. https://www.overleaf.com/