Trabajo Final de Análisis de Lenguajes de Programación

Bruno Baruffaldi

Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniera y Agrimensura

22/12/2017

1. Descripción

Este trabajo consiste en un bot de telegram que brinde información a todos los usuarios de colectivos cuyos viajes tengan origen o destino en la ciudad de Rosario. La principal finalidad de esto es obtener los próximos horarios de colectivos desde la página de la terminal de una manera practica y simple.

2. Esquema de comandos

Los comandos fueron pensados para ser lo mas simples posibles, permitiendo que cualquier usuario sea capaz de comunicarse con el bot de una forma sencilla. Los comandos disponibles son:

- Ayuda: Muestra un mensaje de ayuda, con explicación de los comandos y algunos ejemplos.
- Ciudades: Muestra una lista con todas las ciudades disponibles.
- Viajes a CIUDAD,OPCIÓN: Muestra los próximos viajes desde Rosario a CIUDAD, respetando las opciones que indique el usuario.
- Viajes desde CIUDAD,OPCIÓN: De forma similar al comando anterior, muestra los viajes próximos desde CIUDAD a Rosario.
- Guardar VAR = COMANDO: Permite al usuario almacenar un comando en una variable temporal para facilitar el uso del bot.
- Mostrar VAR: Dada una variable definida, ejecuta el comando que esta almacena y muestra el resultado.
- Queja MENSAJE: Permite al usuario levantar una queja en caso de que el bot tenga algun tipo de error.

Las opciones disponibles son:

1. entre HORA y HORA: Muestra todos los viajes entre el rango horario.
2. ver N: Muestra los proximos N viajes.
3. ver todos: Muestra todos los viajes que son visibles desde la página de la terminal.

En caso de no especificar una opción, se mostraran los próximos 3 viajes por defecto.

3. Módulos del Programa

3.1. BST.hs

- Aquí se define la estructura BST, que a partir de una lista me genera un árbol binario de búsqueda balanceado.
- Las funciones mas importantes que brinda son:

1. *member ::(a→a→Ordering) → BST a → a → Bool*, que dada una función de comparación, un árbol y un elemento determina si dicho elemento forma parte del árbol.
2. *fromlistBy :: (a→a→Ordering) → [a] → BST a*, que dada una función de comparación y una lista, me genera un árbol binario de búsqueda balanceado que respete el orden de la función de comparación.
3. *searchBy :: (a→b→Ordering) → BST b → a → Maybe b*, que dada una función de comparación, un árbol y un elemento, me determina si el elemento o algún valor equivalente están almacenados en el árbol.

3.2. Map.hs

- Su principal función es almacenar pares (clave,valor) de manera tal que no haya dos claves iguales en la estructura.
- La estructura Map esta definida en función de la estructura Red Black Tree para optimizar los tiempos de busqueda e inserción.
- Las principales funciones son:
  1. *emptyMap :: Map a b*, que me permite crear un Map vacío.
  2. *insert :: Ord a ⇒ Map a b → a → b → Map a b*, que dado un Map, una clave y un valor me lo inserta en la estructura evitando almacenar elementos repetidos.
  3. *search :: Ord a ⇒ Map a b → a → Maybe b*, que dada una clave y un map, me devuelve el valor almacenado.
  4. *erase :: Ord a ⇒ Map a b → a → Map a b*, que dada una clave y un map, me permite eliminar dicha clave de la estructura.

3.3. Types.hs

- Este módulo contiene ademas el AST de los comandos y en el se definen los principales tipos de datos:
  - Week = W (Bool,Bool,Bool,Bool,Bool,Bool,Bool,Bool)
  - Info = I (Empresa,Time,Time,Week)
  - Viaje = V (Ciudad,Ciudad,[Info])
  - Empresa = E String
  - Ciudad = C String
  - Time = T (Integer,Integer)
  - Variable = Var String
  - Memoria = Memo (Map Integer (Map Variable Comand))
  - Mensaje = Msm (ChatId,String)
  - Error a = Err String — Result a

3.4. Data.hs

- Almacena un lista con de pares (Ciudad,Código) para hacer posible las consultas en la pagina web de la terminal.
- Contiene funciones para poder manipular algunos tipos de datos que estan definidos en Types.hs
  Algunas de ellas son:
  1. *init_mem :: Memoria*, que representa la memoria vacia.
  2. *lookfor :: ChatId→Variable→Memoria→Error Comand*, que dado un chat, una variable y un entorno me busca la variable en el entorno.
  3. *update :: ChatId → Variable → Comand → Memoria → Memoria*, que dado un chat, una variable y un comando, me lo almacena en la memoria correspondiente.
  4. *printCiudades :: Int → Int → String*, que muestra las ciudades entre los indices numéricos dados.
  5. *codigo :: Ciudad → Maybe Int*, que dado una ciudad me devuelve el código correspondiente.
  6. *isCiudad :: Ciudad → Bool*, que me determina si la ciudad que toma como argumento se encuentra o no en la lista de ciudades disponibles.
  7. *trcd :: Ciudad → Ciudad*, dada una ciudad que pertenece a la lista, pasa una ciudad de minúscula a mayúscula.
  8. *perhaps :: Ciudad→String*, que dada una ciudad me devuelve un mensaje sugiriendo alguna de las ciudades de la lista.

3.5. Parser.hs

- Los parsers son los encargados de tomar los comandos proporcionados al bot y transformarlos en algo con lo que el programa pueda trabajar. En este caso cada vez que el usuario envíe un mensaje al bot, este lo parsea a través de la función readCom.
  readComm :: String → IO (Error Comand) es la funcion principal que dado un String me extrae los comandos que este almacena o me devuelve un error en caso de que la cadena no tenga el formato esperado.

3.6. Telegram.hs

- Este módulo se encarga de enviar y recibir mensajes como cadenas de caracteres utilizando la API de telegram.
- Permite ademas modificar las especificaciones de la función receive.
- Las funciones que este módulo brinda son:
  1. *bot_init :: IO Manager*, que inicializa el bot brindando un elemento de tipo Manager para poder trabajar.
  2. *receive :: Manager → Maybe Int → IO (Error Mensaje,Maybe Int)*, que recibe las actualizaciones a través de la función getUpdates que se encuentra en la API de telegram. Cabe aclarar que nuestro programa esta diseñado para recibir de a una actualización por vez.

3. *send :: Manager → Mensaje → IO (Error MessageResponse)*, envía un mensaje a través de telegram, ya sea para responder a una consulta o reportar un error.

3.7. Main.hs

Es el módulo principal, su funcion es coordinar todos los módulos anteriores y trabajar para responder las consultas que recibe el bot de la forma mas rápida posible, utilizando las funciones definidas en ellos. Para ello utiliza la función *run_bot* y la monada *Mem m a = M {runM::Memoria→m (a,Memoria)}*.

4. Decisiones de diseño

- Cada query que reciba el bot va a tener un numero o id único, el cual se va incrementando a medida que llegan nuevas consultas. Es necesario llevar lastUpdate para no responder dos veces la misma consulta. Con cada consulta que se responde lastUpdate se actualiza, solucionando este problema.
  Por otra parte, las consultas se almacenan en el servidor de telegram por a lo sumo 24 horas, y si el bot no recibe ninguna consulta en 7 días entonces el id de las actualización cambia por un numero random. Es por esto que fue necesario tener en cuenta cuando se recibió la ultima actualización.
  Si no recibi consultas por 24 horas, entonces el buffer del servidor de telegram va a estar vacío y por mas que pida todas las consultas disponibles (lastUpdate=initUpdateId), no voy a recibir ninguna. Si el bot no recibe consultas por 7 días entonces tengo que usar lastUpdate=initUpdateId, pues el nuevo id es asignado de forma aleatoria por telegram.
  Por lo tanto, calculando cuanto tiempo transcurrió desde la ultima actualización podemos setear lastUpdate con el valor initUpdateId evitando responder consultas repetidas sin dejar de lados consultas nuevas que llegue después de un tiempo con un nuevo id.

- En el módulo Html.hs para hacer el codigo mas legible se definió la monada Handler, que en caso de que en algún momento de la ejecución se produce un error, este se propague de forma sencilla.
  En este módulo tambien se implemento la monada Reader y se definieron las funciones next, throw, try y search para poder encontrar información en el codigo html de forma sencilla, una vez que este esté parseado.

- En el módulo Telegram.hs se definieron las funciones send, receive y bot_init para que funcionaran a modo de interfaz y de esta forma simplificar la comunicación entre el programa y telegram.

- Para obtener la lista de las ciudades con sus respectivos códigos se uso el programa Buscador.hs, el cual que utiliza una gran cantidad funciones definidas en Html.hs.

5. Posibles extensiones

- En caso de ser necesario podría implementarse una base de datos con información acerca de los viajes y horarios de la terminal para mejorar los tiempos de respuesta del bot.
- Una de las extensiones mas importantes a corto plazo seria modificar el alcance del bot integrándolo con whatsapp. De esta forma el programa podría llegar a una cantidad de usuarios mucho mayor. Ademas, whatsapp anuncio en septiembre de este año la integración de bots a su plataforma (aunque no de forma gratuita). Mas información en:
  - https://www.adslzone.net/2017/09/01/whatsapp-pymes-soportara-comandos-para-los-bot-como-telegram/
  - http://www.revistagq.com/noticias/tecnologia/articulos/whatsapp-business-asi-funciona/26780

6. Bibliografía

- Se obtuvo información acerca del funcionamiento de la API de telegram desde https://core.telegram.org/bots/api y https://hackage.haskell.org/package/telegram-api
- Se utilizó información de las librerías parsec, bktrees, html-parse, Network.HTTP, Data.Time, telegram-api, entre otras.
- Se utilizó código de https://gist.github.com/owainlewis/4132852
- También se utilizaron funciones definidas para el primer trabajo practico de la materia Análisis de Lenguajes de Programación y se siguió el esquema de uno de sus archivos.