UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA

LABORATORIO 1

PARADIGMAS DE PROGRAMACIÓN

Alberto Rodríguez Z.

Profesores:	Roberto González
	Daniel Gacitúa
	Víctor Flores
Fecha de Entrega:	23-04-2018

Santiago de Chile

1- 2018

TABLA DE CONTENIDOS

TABLA DE FIGURAS

CAPÍTULO 1. INTRODUCCIÓN

El presente informe tiene como objetivo principal ser una referencia a la línea de pensamiento y al contexto de desarrollo del código fuente que lo acompaña para la presentación del laboratorio 1 del curso de Paradigmas de Programación. En esta ocasión se hará uso del paradigma de programación Funcional con el lenguaje de programación Scheme.

En primer lugar, el problema a resolver es la creación e implementación de un chatbot. De este mismo modo, es relevante mencionar que un chatbot se puede definir como programas computacionales que pueden mantener una conversación con un ser humano, sin embargo, esta interacción queda limitada por un contexto/temática y una serie de parámetros. Asimismo, los chatbot resultan ser de utilidad para manejar algunas fases de la conversación como pueden ser saludos de bienvenida, solicitud de datos y responder a algunas preguntas específicas hechas por el usuario.

Por consiguiente, en este informe la solución implementada para el chatbot trabaja en base a la temática de atención al cliente en un negocio de comida rápida, en donde se crean flujos conversacionales protocolares, estructuras gramaticales simples y vocabulario conocido por el chatbot. De tal forma dando los parámetros y temática se puede mantener una conversación con el chatbot.

Por lo tanto, se establece como objetivo de este laboratorio, la aplicación de las técnicas aprendidas en clases, además ir más allá e investigar cuando corresponda, con el fin de crear un algoritmo eficaz que funcione bajo el paradigma funcional.

CAPITULO 2. MARCO TEÓRICO

2.1 PARADIGMA FUNCIONAL

Como su nombre lo indica, este paradigma se basa fuertemente en el concepto de función, en este caso basado en el uso de funciones matemáticas, a diferencia de la programación imperativa que le da importancia a los cambios de estado mediante mutación de variables. Una de las diferencias más importantes con respecto a otros paradigmas es que solamente trabajaran con sus valores de entradas y con constantes predefinidas, en este paradigma no existe la definición de variables, es decir, no puedes cambiar ni modificar algún dato como también la falta de construcciones estructuradas como la secuencia o la iteración (lo que obliga en la práctica a que todas las repeticiones de instrucciones se lleven a cabo de funciones recursivas). Básicamente, la programación funcional son solamente una composición de funciones que trabajan limitadamente con cada elemento que se le pase como argumento con algún valor.

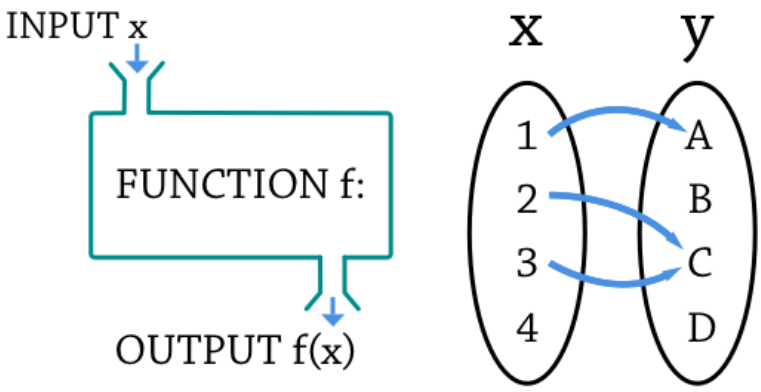

Figura 1.- Diagrama de funcionamiento de un paradigma funcional que se asemeja a las funciones matemáticas

La programación funcional tiene transparencia referencial esto quiere decir que para un valor de entrada produce siempre la misma salida. En temas de elegancia, legibilidad y flexibilidad los programas funcionales suelen ser más claros, concisos y bellos, si estas leyendo una función, solo te enfoques en esa función en sus entradas y salidas. Otra de las ventajas de la programación funcional es que existe mayor facilidad para las pruebas del programa y la depuración.

CAPÍTULO 3. DESCRIPCIÓN DEL PROBLEMA

Se solicita crear un algoritmo en el lenguaje de programación Scheme que simule la conversación entre el usuario y un chatbot, un chatbot que contiene personalidad, vocabulario y evaluaciones de diálogo. La conversación debe funcionar en base a una temática manteniendo una conversación fluida con un inicio de la conversación y un final. La conversación se va almacenando en un registro llamado log como también existe la semilla seed que normará el comportamiento de las funciones ante funciones pseudoaleatorias. El algoritmo debe poder efectuar las siguientes funciones:

1.**beginDialog**: Permite iniciar una nueva conversación con el chatbot donde el mensaje inicial es ofrecido por el chatbot.
2.**sendMessage**: Permite realizar envío de los mensajes del usuario al chatbot, donde este debe poder generar una respuesta a este mensaje.
3.**endDialog:** Permite dar cierre a la conversación donde el mensaje final es enviado por el chatbot.
4.**rate:** Permite evaluar el desempeño del chatbot frente a las conversaciones, se realiza la evaluación del usuario y la autoevaluación del chatbot.
5.**test:** Permite ilustrar el desarrollo de una conversación entre un usuario y un chatbot. Este usuario en una lista de strings que corresponder a los mensajes enviados por el usuario

Además de estas funciones, se deben implementar abstracciones de una estructura TDA de 6 capas y se debe representar e implementar el TDA en las funciones del código sin perjudicar la eficiencia de este.

Todas estas funciones tienen un dominio y un recorrido definidos para su desarrollo, aunque se tiene libertad en algunos casos del orden o que funciones a parte de estas se trabajen en ellas.

3.1 ANÁLISIS

El principal problema radica en la capacidad de darle “sentido” a una conversación, ya que cuando el usuario envía un mensaje, el chatbot debe ser capaz de responderle lógicamente en relación con lo preguntado por este. Por lo tanto, debe existir una gran posibilidad de respuestas para que el chatbot tenga la capacidad de seleccionar la indicada.

De otro modo, uno de los puntos complejos de trabajar en Scheme es que al no contar con las instrucciones While y For, no se puede iterar la cantidad de respuestas que pueden existir, la solución se debe desarrollar únicamente en funciones recursivas y saber trabajar sobre las listas de Scheme.

CAPÍTULO 4. DESCRIPCIÓN DE LA SOLUCIÓN

Puesto que hay funciones y parámetros definidos, el desafío se centra en poder trabajar estas funciones entre sí y cómo el recorrido de una sirva de dominio de otra. Siguiendo el orden de parámetros se define primero la estructura chatbot como una lista de listas, cada una de estas listas define un parámetro del chatbot, que serían la personalidad, vocabulario y evaluaciones de diálogo.

El chatbot contiene 3 personalidades que son amable, animado y agresivo. Las respuestas que entregará dependerán de la personalidad que tenga el chatbot, la cual se elige a través de la semilla seed. Este último corresponde a un número, el cual elige aleatoriamente que personalidad tendrá el chatbot ante cierta conversación con el usuario. Dado esto, para que el chatbot mantenga la misma personalidad a través de la conversación se debe ingresar siempre la misma semilla o si no variará la personalidad del chatbot.

Para que del número ingresado como semilla se obtenga una personalidad para el chatbot, se crea una función random que retorna un número aleatorio. Luego, a partir de este se ingresa en la función "semillaDeDecisiones" que retorna un número que puede ser 1,2 o 3, los cuales cada uno representa una personalidad especifica del chatbot (1: amable, 2: animado, 3:agresivo).

```
(define (semillaDeDecisiones semilla)
  (cond ((= (remainder (myRandom semilla) 3) 0) 3)   ; cada resto te lleva a una
        ((= (remainder (myRandom semilla) 2) 0) 1)       ; personalidad distinta
        ((= (remainder (myRandom semilla) 2) 1) 2)))
```

También, con el seed obtendremos con qué personalidad nos saludará y se despedirá el chatbot. El saludo dependerá de la hora, es decir, si se realiza en la mañana, tarde o noche. El código se desarrolla en el IDE Dr.Racket por lo que podremos importar la librería de Scheme (srfi) de donde se importa srfi/19 y srfi/27 que gracias a ellas se puede obtener la fecha y hora exacta a la que el programa es ejecutado, determinado así a qué hora se puede hablar con el chatbot.

De otro modo, ya que en Scheme no existen las iteraciones se necesitaba poder enviar la información de una función a otra, por lo que se trabaja con el TDA de listas de Scheme, ya que tienen la implementación y representación necesaria para poder solucionar el problema. Por otra parte, el log, que es el registro histórico de conversaciones, se trabaja como lista, ya que se ingresará cada conversación como String dependiendo de que función sea.

Comenzando con la función “BeginDialog” se ingresa el chatbot, el log que inicialmente es una lista vacía y la semilla seed, para después comprobar a través de las capas del TDA si cumple con la estructura y se transforma la respuesta del chatbot como un String y se guarda como primer elemento en el log. Lo mismo ocurre con la función “endDialog” que se trabaja igual que la función anterior solo que se agrega como último elemento de la lista log.

Para desarrollar la función “sendMessage” es totalmente distinto a las anteriores porque además se ingresa el parámetro mensaje, que es el mensaje (string) enviado por el usuario, por lo que se debe crear una función capaz de verificar que deba responder el chatbot ante cierto mensaje.

Después que el usuario ingresa un mensaje, el cual representa el dominio junto a los parámetros log y seed de la función “respuestaUsr”, en donde se verifica qué responder. Esta función consta de la instrucción cond en donde contiene las posibilidades de respuestas. El mensaje del usuario se transforma en una lista con cada palabra como elemento (en esta función ya ingresa como lista) y se ingresa en cada condición que contiene una cierta cantidad de strings. Si esos strings se encuentran cómo algunos elementos de la lista mensaje el chatbot va a tener una respuesta determinada, así se puede crear una coherencia de que puede responder el chatbot comparándolo con los strings.

Para poder comparar estos Strings se necesita poder recorrer la lista, para esto usaremos la recursión de cola que retornará a la función, pero con la lista modificada eliminando los elementos que no cumplen con las condiciones. Si se necesita eliminar algún elemento de una posición determinada de la lista se crea la función “eliminarElemento”, que utiliza la recursión natural al eliminar el elemento que se busca y retorna la lista modificada.

```
(define (eliminarElemento elem list)
   (if (equal? elem (car list))
     (cdr list)                                          ;Recursión natural: queda como estado
     (cons (car list) (eliminarElemento elem (cdr list)))))  ;pendiente el car de la lista
```

Después de encontrar la respuesta correcta para el mensaje del usuario se guarda en el log el mensaje del usuario y la respuesta del chatbot, y así se puede generar un flujo de conversación entre el usuario y el chatbot siempre en el contexto de tienda de comida rápida.
La conversación fluye normal hasta que se llama a la función “endDialog” donde finaliza la conversación y al final el chatbot autoevalúa la conversación que se obtiene a través de la función “rate” que entrega los resultados de las evaluaciones de las conversaciones.

Para finalizar, la descripción de la solución también existe la función test que simula una conversación completa de inicio a final, se ingresan los mismos parámetros, pero el mensaje ya no es un string más bien una lista que contiene como elemento los mensajes de la conversación (strings) y cada uno es la respuesta siguiente del usuario. Se aplica recursión natural a la función "test" pero con el log modificado donde se va guardando toda la conversación.

```
(define (test user chatbot log seed)
  (if  (null? log)
     (test user chatbot (beginDialog chatbot log seed) seed)   ;La funcion llama a todas las demás
     (if (null? user)                                          ;"beginDialog" "endDialog" "sendMessage"
        (endDialog chatbot log seed)                           ;asi puede desarrollar la conversacion completa
        (test (cdr user) chatbot (sendMessage (car user) chatbot log seed) seed))
     )
  )
```

CAPÍTULO 5. RESULTADOS

Tiempos de respuesta aceptable, es decir, el usuario no nota o queda en espera de resultados, ya sea cuando envía el mensaje que el chatbot tiene que responder recorriendo las variadas posibilidades para verificar, además de estar usando funciones recursivas que utiliza un excesivo uso de memoria para ejecutarse podría tardarse mucho más del tiempo de espera.

Como se muestra en la figura 2, en primera instancia se muestra el log que contiene la conversación entre el usuario y el chatbot.

```
> log1
(("BeginDialog a las [23/04/2018 04:42:35]-Chatbot: Buenas noches bienvenido a Mr. Ham que desea ordenar?"
  "[23/04/2018 04:43:24]-Usuario: hola, cual es tu nombre"
  "[23/04/2018 04:43:24]-Chatbot: Mi nombre es Alberto"
  "[23/04/2018 04:44:15]-Usuario: bueno que hamburguesas tiene"
  "[23/04/2018 04:44:15]-Chatbot: hay hamburguesa clasica,vegetariana y gigante"
  "[23/04/2018 04:44:49]-Usuario: ya quiero una hamburguesa vegetariana"
  "[23/04/2018 04:44:49]-Chatbot: Bueno quiere algo mas?"
  "[23/04/2018 04:45:36]-Usuario: no nada"
  "[23/04/2018 04:45:36]-Chatbot: Bueno, anotado todo"
  "EndDialog a las [23/04/2018 04:45:59]-Chatbot: Hasta luego gracias por su pedido"))
```

Figura 2.- conversación con entre el usuario y el chatbot

Así se obtiene los datos de una conversación con una fecha determinada, también se puede ver en la figura 3 cuál es el recorrido de la función test.

```
> (test user2 chatbot '() 8)
(("BeginDialog a las [23/04/2018 04:55:03]-Chatbot: Hola muy buenas noches mi estimado bienvenido a Mr. Ham en que lo puedo ayudar?"
  "[23/04/2018 04:55:03]-Usuario: como te llamas"
  "[23/04/2018 04:55:03]-Chatbot: No me lo habian preguntado antes gracias!, me llamo Alberto"
  "[23/04/2018 04:55:03]-Usuario: ya bueno, me gustaria una hamburguesa"
  "[23/04/2018 04:55:03]-Chatbot: Tenemos de muchos tipos cual desea?"
  "[23/04/2018 04:55:03]-Usuario: que pesado como te sientes ?"
  "[23/04/2018 04:55:03]-Chatbot: Animado"
  "[23/04/2018 04:55:03]-Usuario: asi veo, quiero una bebida pepsi"
  "[23/04/2018 04:55:03]-Chatbot: rica bebida! algo mas que quiera?"
  "[23/04/2018 04:55:03]-Usuario: no"
  "[23/04/2018 04:55:03]-Chatbot: oh bueno, espero que lo disfrute"
  "EndDialog a las [23/04/2018 04:55:03]-Chatbot: Suerte en todo y gracias por su pedido"))
>
```

Figura 3.- desarrollo función test

CAPÍTULO 6. CONCLUSIONES

En primer lugar, el paradigma de programación funcional tiene la ventaja de ser más elegante, legible y flexible. Además, los programas suelen ser más claros y más concisos, siendo así más fáciles de evaluar, dado que solamente se enfocan en una función en específico. Por otra parte, si quieres verificar errores de algún tipo de cosa se puede realizar fácilmente el seguimiento de flujo del programa. De otro modo, debido a su estructura es posible organizar de mejor manera el programa. Por otro lado, es casi natural poder dividir el problema en unos de menos escala.

En segundo lugar, respecto al problema en sí del laboratorio, ha sido interesante plantear el problema de cómo poder desarrollar una conversación con un chatbot, a través de las distintas funciones y técnicas que tiene el paradigma funciones, junto con las implementaciones que trae Scheme. De este modo, se puede afirmar que se realizó un gran trabajo con Scheme al poder trabajar directamente con el TDA de las listas y aplicando recursión.

Finalmente, puesto que se ha logrado desarrollar un chatbot con una cierta temática, con el cual uno puede mantener una conversación, es posible concluir que se ha logrado cumplir con el objetivo principal de este laboratorio.

BIBLIOGRAFÍA

•https://www.gnu.org/software/guile/manual/html_node/R6RS-Libraries.html#R6RS-Libraries

•http://www.larcenists.org/Documentation/Documentation0.99/user-manual.chunked/ar01s04.html

•https://srfi.schemers.org/srfi-19/srfi-19.html

•https://docs.racket-lang.org/reference/define-struct.html?q=define-strutc#%28form._%28%28lib._racket%2Fprivate%2Fbase..rkt%29._define-struct%29%29

• https://es.linkedin.com/pulse/breve-introducci%C3%B3n-al-paradigma-funcional-emanuel-casco

•https://docs.racket-lang.org/reference/pairs.html?q=pair%3F#%28def._%28%28quote._~23~25kernel%29._pair~3f%29%29