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

Laboratorio 1 - Paradigma Funcional
Sistema de chatbots simplificado
Racket - Scheme

Nombre: Vicente Gerardo Arce Palacios
Curso: Paradigmas de Programación
Sección A-1
Profesor: Roberto González Ibañez

2 de Octubre de 2023

Tabla de contenidos

1. Introducción

En el presente informe se abordará el problema propuesto en el enunciado del laboratorio 1 del ramo "Paradigmas de Programación". En este primer laboratorio se tiene como requisito el uso del Paradigma Funcional, en donde se utilizó Racket-Scheme como lenguaje de programación. El informe se separara en una breve descripción del problema y del paradigma a utilizar, más adelante en el desarrollo se detallara el análisis del problema, el diseño de la solución, algunos detalles en la implementación junto con las instrucciones de uso, se darán los resultados y una autoevaluación para finalmente dar una conclusión al proyecto.

1.1. Descripción del problema

Se presenta como desafío la creación de un sistema el cual pueda albergar chatbots. Este sistema permite funcionalidades tales como añadir chatbots, registrar usuarios, loguear usuarios y desloguear usuarios, a su vez el usuario tendrá la posibilidad de interactuar con el sistema contenedor de chatbots, esto a través de la consola de ´´Dr.Racket", esto se puede lograr gracias a funciones como system-talk-rec o system-talk-norec. A modo de sintetizar, se podría decir que el problema se reduce a explorar y pensar en como se puede aplicar el paradigma funcional en el lenguaje de programación Racket para la creación de un sistema de chatbots.

1.2. Descripción del paradigma

Para saber que es el paradigma funcional, se debe saber que es un paradigma que proviene de una familia de paradigmas declarativos. La programación funcional, al contrario de otros paradigmas como podrían ser la programación imperativa, se basa en escribir código que especifique el "¿Qué?" se quiere hacer, sin especificar el "¿Como?" se podría llegar a realizar, además, otra cosa que se destaca de este paradigma es la no mutabilidad de los datos.
El paradigma de la programación funcional usa el calculo lambda (funciones anónimas) y tiene como unidad de abstracción las funciones, es decir, en este paradigma se considera todo como una función. Una función corresponde a una transformación de elementos de un dominio, para a través del proceso de transformación, dar entrega del recorrido.
La recursividad es una de las técnicas usadas en la programación funcional, existen varios tipos de recursión, entre ellos están, la recursividad natural, que tiene como característica el no dejar estados pendientes al momento de la realización de la recursividad, la recursividad de cola, la cual se caracteriza por no dejar estados pendientes,por ultimo esta la recursión arbórea, la cual se basa en una función que se llama múltiples veces a si misma.

2. Desarrollo

En este apartado se abordara el análisis del problema, el diseño de la solución, aspectos en la implementación, las instrucciones de uso junto con los resultados y una autoevaluación, además, como ya se menciono en la descripción del paradigma, este trabaja con la no mutabilidad de los datos, es por esto que cuando a lo largo de el presente se use la terminología "modificar" en realidad se hace referencia a una "reconstrucción" de algún o algunos datos.

2.1. Análisis del problema

Primeramente, dados los requisitos funcionales que se deben realizar, lo primero que se debe hacer es identificar los TDA necesarios para la resolución del problema dado. Se ve que el TDA principal es el TDA sistema, el cual es quien guarda los chatbots, usuarios registrados y logueados junto con el chat-history de cada usuario. El sistema, por los requisitos funcionales pedidos, tiene la capacidad de registrar y actualizar el historial de los usuarios. Otro aspecto a tener en cuenta es que los chatbots contienen flujos que a su vez contienen opciones, lo cual puede derivar a pensar en estos como nuevos TDA.
El laboratorio al estar pensado en un lenguaje basado en listas como Racket/Scheme, el TDA System debe ser una lista compuesta por atributos como un nombre, código de chatbot inicial, chatbots, usuarios registrados/logueados, un chatHistory y por ultimo los códigos de chatbot y flujo actuales. Las funciones **system-add-user**, **system-login**,**system-logout**, son funciones que permiten añadir usuarios al sistema, logueralos y desloguearlos respectivamente, también las funciones **system-talk-rec** y **system-talk-norec** son funcionalidades las cuales pueden ser usadas por el usuario para interactuar con el sistema, estas funciones tienen las capacidad de guardar el historial y modificar los códigos de chatbot y flujos actuales en el sistema, para finalmente retornar un sistema pero con la modificación de lo anteriormente mencionado, a su vez, el historial puede ser mostrado por el uso de la función **system-synthesis**. Todas las funcionalidades anteriormente mencionadas, permiten la interacción o modificación del sistema de alguna forma o con algún propósito especifico, pero para lograr implementar estas mismas se debe pensar en como se deben implementar los TDA que componen el sistema, como ya se menciono, un sistema contiene chatbots, los chatbots contienen flujos y los flujos contienen opciones. Lo anteriormente descrito se puede pensar como un árbol, en donde la **raíz** sería el sistema que como hijos están los chatbots, chatbots que como hijos tienen flujos y flujos que como hijos tienen opciones, opciones que serian las **hojas**, cada uno de estos TDA que se podrían pensar como parte de un gran árbol fueron implementados con la finalidad de poder suplir los requisitos funcionales, el como fue esta implementación se abordara en el diseño de la solución.

2.2. Diseño de la solución

Para llegar al diseño de la solución, se hizo un diagrama, el cual puede ser encontrado en el Anexo 1, diagrama el cual tiene una forma de un árbol como se menciono en el análisis del problema. A continuación se especificaran los TDA más importantes.

2.2.1. TDA

Los TDA presentados a continuación cuentan con funciones constructoras, selectoras, modificadores y otras, según las necesidad de estas para la resolución del problema.

TDA System: Lista de [name(string) X InitialChatbotCodeLink(integer) X chatbots (lista de chatbots) X '() (lista de historial) X '() (usuarios registrados) X '() (usuarios logueados) X actualChatbotCodeLink (integer) X actualFlowCodeLink (integer)].

TDA Chatbot: Lista de [chatbotID(integer) X name(string) X welcomeMessage (string) X startFlowID(integer) X flows(lista de flujos del chatbot)].

TDA Flow: Lista de [id(integer) X name-msg(string) X Option(lista de opciones)].

TDA Option: Lista de [code(integer) X message(string) X chatbotCodeLink(integer) X InitialFlowCodeLink(integer) X Keyword (lista de palabras claves)].

2.2.2. Diseño de la solución y TDA

Los TDA anteriormente mencionados son los más importantes para llegar a la solución del problema dado, si bien se implementaron los TDA user y chatHistory, su implementación es más básica que la de los demás ya descritos.
El TDA system, es el más importante, ya que en el se encuentran los demás TDA, se decidió añadir un actualChatbotCodeLink y actualFlowCodeLink al sistema, esto pensando en los requerimientos funcionales 12, 13 y 15, ya que es primordial saber en que chatbot y flujo se encuentra el sistema e ir modificándolo a medida que se ocupen dichas funciones. Para la función 9, la segunda lista del sistema es primordial, ya que ahí es donde se añaden los usuarios registrados y además en la primera lista es donde se crea un historial para cada usuario que se registre, para la función 10, los usuarios que están en la segunda lista del sistema son los que podrán loguearse en el sistema, es decir, se agrega a la tercera lista, para la función 13 simplemente se vacía la tercera lista si hay usuarios logueados, para la función 14, se busca un usuario dado en la primera lista y se retorna su historial asociado.
El resto de requisitos funcionales los cuales son constructores y modificadores de los TDA mencionados se encuentran implementados, pero no se considera relevante el como se diseñaron debido a la sencillez de su realización.

2.3. Aspectos de implementación

La realización de este proyecto fue implementada en el lenguaje de programación Racket/Scheme utilizándose el compilador de Dr.Racket 8.10.
El proyecto cuenta con cada función de los requerimientos funcionales documentada con su nombre, dominio, recorrido, recursión y una breve descripción. El proyecto se encuentra dividido en 7 archivos más un script de pruebas/ejemplo.
Ver Anexo 2 para ver en que archivos se encuentra cada requerimiento funcional.

2.4. Instrucciones de uso

Primeramente, se debe verificar que se tengan todos los archivos en una misma carpeta, una vez realizado esto, en el archivo script de pruebas, existen varios ejemplos para cada una de las operaciones, dicho script al ser ejecutado a través de la opción "Run" (Algunos de los ejemplos se pueden visualizar en el Anexo 3).
En el caso de querer crear un nuevo sistema, dejando de lado el de prueba, se deben seguir los siguientes pasos:
1) Se deben crear las opciones, esto a través de su función constructora "option".
2) Añadir a cada flujo las opciones correspondientes, esto se hace con la función "flow".
3) Se encapsulan los flujos en los chatbots, esto con el constructor de chatbots "chatbot".
4) Se incluyen los chatbots a un nuevo sistema con el constructor de sistemas "system".
5) Se registran los usuarios con la función "system-add-user", se loguean con la función "system-login". Una vez logueado el usuario, con la función "system-talk-rec" o "system-talk-norec" se puede interactuar con el sistema de chatbots. Para ver el historial de un usuario en particular, se puede realizar con la función "system-synthesis". Finalmente con la función "system-logout" se puede desloguear al usuario.
6) Otra alternativa al paso anterior es usar la función "system-simulate", la cual realizara un simulación de una interacción con un sistema de chatbots ya creado.

Un fallo que se puede llegar a generar es cuando se interactúa con el sistema, ya que si bien se soporta el case sensitive, si se coloca una opción o keyword que no esta en el chatbot y flujo especifico, el sistema se caerá. Al hacer pruebas del programa no se encontraron más errores, funcionando correctamente todas las funcionalidades.

2.5. Resultados y autoevaluación

El grado de alcance de los requisitos funcionales fue logrado en su totalidad, contándose con todos los requerimientos funcionales y no funcionales. Se lograron los resultados previstos.
La autoevaluación puede ser encontrada en el Anexo 4, con un puntaje siguiendo el siguiente formato: 0: No realizado/ 0.25: Funciona el 25 % de las veces/ 0.5: Funciona el 50 % de las veces/ 0.75: Funciona el 75 % de las veces/ 1: Funciona el 100 % de las veces.

3. Conclusión

Luego de la realización del proyecto, se puede llegar a la conclusión de que se cumplió el objetivo de la realización de un sistema de chatbots utilizando el lenguaje Racket/Scheme, es decir, siguiendo el paradigma de la programación funcional del cual se adquirió el aprendizaje necesario sobre el paradigma para llevar a cabo el proyecto, a su vez, fue todo un desafío iniciar este proyecto, debido a la costumbre de pensar de manera imperativa.
Una de las complicaciones que se tuvieron al momento de la realización del proyecto fue la no existencia de variables, lo cual conllevaba a que muchas de las ideas en un principio no tuvieran sentido en este paradigma, lo anteriormente mencionado se puede ver reflejado en la extensión y lo engorrosas que son las funciones 12 y 13, que probablemente existe una forma bastante más simple de llevar a cabo dichas funcionalidades.
Para sintetizar, dejando de lado las complicaciones que hubieron, gracias a este proyecto se pudo aprender sobre muchas herramientas como el uso de Git, GitHub y lenguaje Racket/Scheme. Lo más importante es que en este laboratorio se aprendió a pensar de manera funcional. Queda como una autocrítica el poder adaptarse de una manera más rápida a un nuevo paradigma y lograr salir de la zona de confort de la programación imperativa.

4. Bibliografía

Gallardo, D., Martínez, F., Botía, A., & Pomares, C. (2022). Tema 2: Programación funcional. Tema 2: Programación funcional - LPP. https://domingogallardo.github.io/apuntes-lpp/teoria/tema02-programacion-funcional/tema02-programacion-funcional.html

Chauhan, A. (2023, September 23). Abstract data types. GeeksforGeeks. https://www.geeksforgeeks.org/abstract-data-types/

Flatt, M., & Findler, R. B. (2023). The Racket Guide. https://docs.racket-lang.org/guide/

5. Anexos

5.1. Anexo 1: Diagrama del sistema

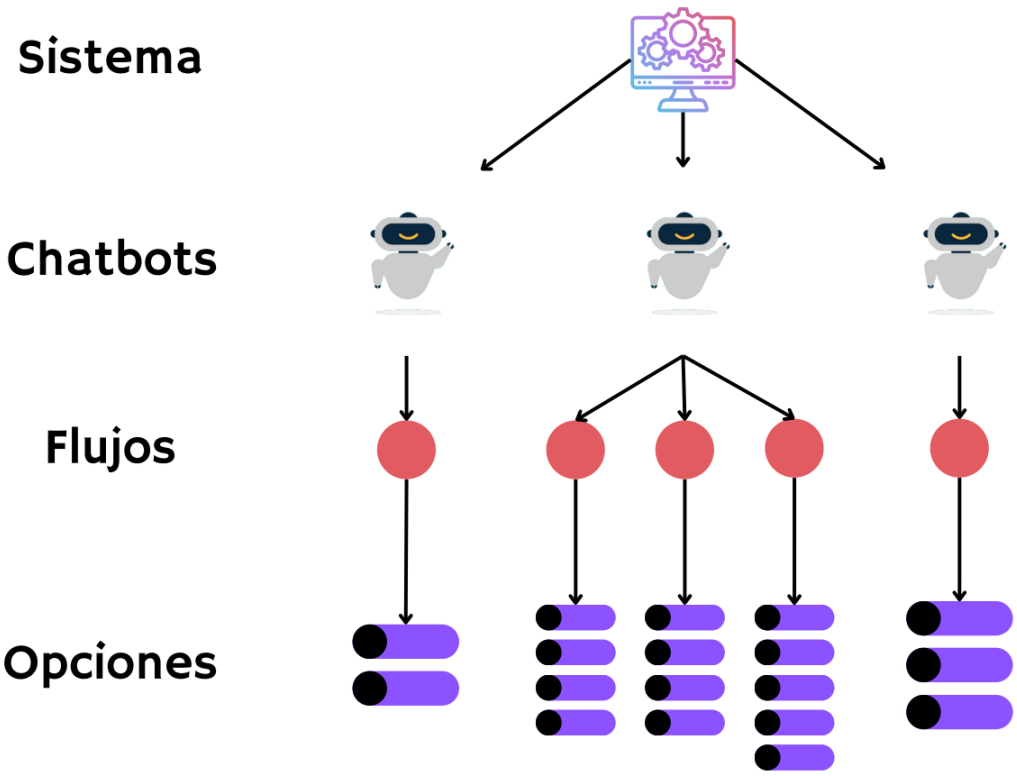

Figura 1: Diagrama de abstracción del sistema.

5.2. Anexo 2: Ubicación de las funciones

Nombre de archivo	TDA_Option	TDA_Flow	TDA_chatbot	TDA_system	TDA_chatHistory	TDA_user	Main
Constructores	option	flow	chatbot	system	chatHistory	make-user	
Selectores	get-code-option	get-id-flow	get-chatbotID-chatbot	get-name-system	get-user-chatHistory		
	get-message-option	get-name-msg-flow	get-name-chatbot	get-InitialChatbotCodeLink-system	get-Historial-chatHistory		
	get-ChatbotCodeLink-option	get-Option-flow	get-welcomeMessage-chatbot	get-chatbots-system			
	get-InitialFlowCodeLink-option		get-startFlowId-chatbot	get-chatHistory-system			
	get-Keywords-option		get-flows-chatbot	get-users-system			
				get-logUsers-system			
				get-actualChatbotCodeLink-system			
				get-actualFlowCodeLink-system			
Requisitos Funcionales	option	flow	chatbot	system			system-talk-rec
		flow-add-option	chatbot-add-flow	system-add-chatbot			system-talk-norec
				system-add-user			system-synthesis
				system-login			system-simulate
				system-logout			

Figura 2: Ubicación de funciones creadas.

5.3. Anexo 3: Ejemplos

```
Welcome to DrRacket, version 8.10 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> ;Chatbot 0
(define op1 (option  1 "1) Futbol" 1 1 "futbol" "messi" "Ronaldo")) ;Creando opciones
(define op2 (option  2 "2) basketball" 2 1 "basketball" "Jordan" "Lebron"))
(define f1 (flow 1 "Flujo Principal Chatbot 0\nBienvenido\n¿Cuál es tu deporte favorito?" op1 op2 op2 op1)) ;Creando flujo
(define cb0 (chatbot 0 "Inicial" "Bienvenido\n¿Cuál es tu deporte favorito?" 1 f1 f1 f1 f1))  ;Creando chatbot 0
;Chatbot 1
(define op3 (option  1 "1) Messi" 1 2 "Argentino" "Messi"))
(define op4 (option  2 "2) Cristiano Ronaldo" 1 2 "Ronaldo"))
(define op5 (option  3 "3) Cruyf" 1 2 "Holandes"))
(define op6 (option  4 "4) En realidad no me gusta el futbol" 0 1 "volver"))
(define f2 (flow 1 "Flujo 1 Chatbot1\n¿Quien es tu futbolista favorito?" op3 op4 op5 op6))
(define op7 (option 1 "1) Si le he visto jugar" 1 2 "Si"))
(define op8 (option 2 "2) No lo he visto jugar" 1 2 "no"))
(define op9 (option 3 "3) En realidad me gusta otro jugador" 1 1 "volver"))
(define f3 (flow 2 "Flujo 2 Chatbot1\n¿Le has visto jugar?" op7 op8 op9))
(define cb1 (chatbot 1 "Futbolero" "Bienvenido\n¿Te gusta el futbol?" 1 f2 f3))  ;Creando chatbot 1
;Chatbot 2
(define op10 (option  1 "1) Michael Jordan" 2 2 "Jordan"))
(define op11 (option  2 "2) Lebron James" 2 2 "James" "lebron"))
(define op12 (option  3 "3) Shaquille o'neal" 1 2 "Shaquille" "Shaquile"))
(define op13 (option  4 "4) En realidad no me gusta el basket" 0 1 "volver"))
(define f4 (flow 1 "Flujo 1 Chatbot2\n¿Quien es tu futbolista favorito?" op10 op11 op12 op13))
(define op14 (option 1 "1) Si le he visto jugar" 2 2 "Si"))
(define op15 (option 2 "2) No lo he visto jugar" 2 2 "no"))
(define op16 (option 3 "3) quiero volver atras" 2 1 "volver"))
(define f5 (flow 2 "Flujo 2 Chatbot2\n¿Le has visto jugar?" op14 op15 op16))
(define cb2 (chatbot 2 "Basketbolero" "Bienvenido\n¿Te gusta el basket?" 1 f4 f5));Creando chatbot 2
;Creando sistema
(define s1 (system "Chatbots Paradigmas" 0 cb0 cb0 cb0 cb1 cb2)) ;Creando sistema
(define s2 (system-add-chatbot s1 cb0)) ;igual a s1
(define s3 (system-add-user s2 "user2"));Añadiendo/Registrando usuario
(define s4 (system-login s3 "user2"))   ;Logueando usuario
(define s5 (system-logout s4))          ;deslogueando usuario
(define s6 (system-login s5 "user2"))   ;Logueando usuario
```

Figura 3: Ejemplo creación de sistema.

```
> (define s7 (system-talk-norec s6 "hola"))

> (define s8 (system-talk-norec s7 "1"))
> (define s9 (system-talk-norec s8 "messi"))
> (define s10 (system-talk-norec s9 "3"))
> (display (system-synthesis s10 "user2"))
1696193485-user2: hola
1696193485-Inicial: Flujo Principal Chatbot 0
Bienvenido
¿Cuál es tu deporte favorito?:
 1) Futbol
 2) basketball


1696193492-user2: 1
1696193492-Futbolero: Flujo 1 Chatbot1
¿Quien es tu futbolista favorito?:
 1) Messi
 2) Cristiano Ronaldo
 3) Cruyf
 4) En realidad no me gusta el futbol


1696193497-user2: messi
1696193497-Futbolero: Flujo 2 Chatbot1
¿Le has visto jugar?:
 1) Si le he visto jugar
 2) No lo he visto jugar
 3) En realidad me gusta otro jugador


1696193506-user2: 3
1696193506-Futbolero: Flujo 1 Chatbot1
¿Quien es tu futbolista favorito?:
 1) Messi
 2) Cristiano Ronaldo
 3) Cruyf
 4) En realidad no me gusta el futbol
```

Figura 4: Ejemplo interacción con un sistema.

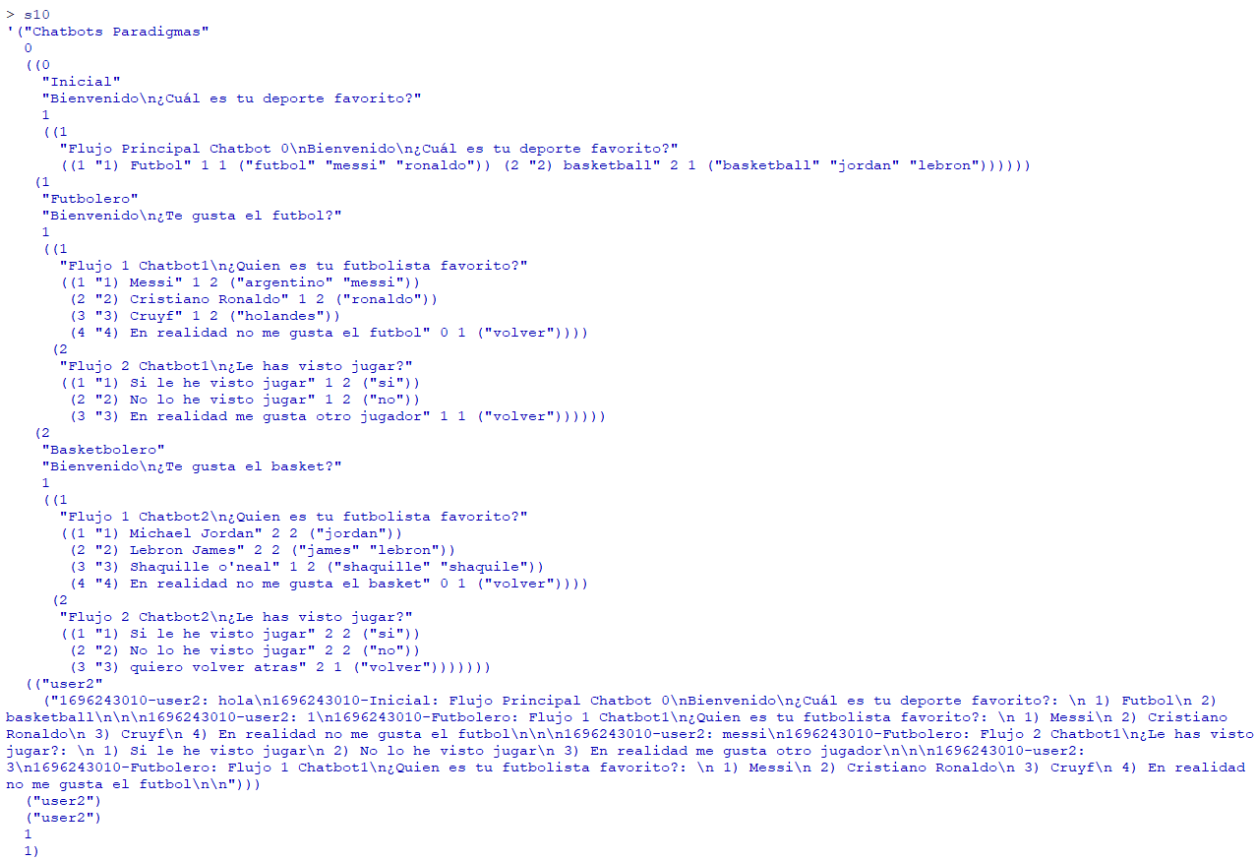

Figura 5: Estructura del sistema s10

5.4. Anexo 4: Autoevaluación

Requirimientos Funcionales	Evaluación
TDAs	1
option	1
flow	1
flow-add-option	1
chatbot	1
chatbot-add-flow	1
system	1
system-add-chatbot	1
system-add-user	1
system-login	1
system-logout	1
system-talk-rec	1
system-talk-norec	1
system-synthesis	1
system-simulate	1

Figura 6: Autoevaluación Requisitos funcionales.