DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Paradigmas de programación Laboratorio 1 (Paradigma Funcional – Lenguaje Scheme)

Estudiante: Elías Zúñiga Tobar

Sección: B-2

Profesor: Gonzalo Martínez

Fecha: 9 de octubre de 2023

Índice

1. Portada……1
2. Índice……..2
3. Introducción……..3
3.1. Problema……..3
3.2. Paradigma……3
4. Desarrollo……..4
4.1. Análisis del problema……4
4.2. Diseño de solución……4
4.3. Aspectos de implementación……5
4.4. Instrucciones de uso……..5
4.5. Resultados……..5
4.6. Autoevaluación……6
5. Conclusiones……...6
6. Bibliografía……7
7. Anexos……7

Introducción

Este informe tiene como objetivo el de aprender, entender y explicar sobre el uso del paradigma funcional a través del lenguaje de programación “Scheme” usando el programa de “DrRacket” para el desarrollo de este laboratorio, en donde se tomarán conceptos que no se aplican en otros paradigmas como en el imperativo.

Problema: En este trabajo se pide el desarrollo e implementación de un sistema TDA para la creación, despliegue y administración de chatbots simplificado, donde un usuario realizar operaciones tales como crear chatbots, interactuar con ellos y ofrecer una síntesis de las interacciones con el bot. Para ello, se necesitará los siguientes elementos:

- **Sistema (System):** Elemento indispensable para la implementación de la solución, se usará como base en donde se agregarán y guardarán los chatbots y los usuarios registrados en el sistema junto con su historial de chat y también su fecha de creación.
- **Chatbot:** Corresponde a los chatbots que se agregarán al sistema, los cuales tendrán una ID, un mensaje entrante, y flujos.
- **Flujo (Flow):** Los flujos serán aquellos elementos que guardarán las opciones, estos contarán también de una ID, un mensaje entrante y las opciones.
- **Opciones (Option):** Estos serán las opciones que el usuario tendrá que escoger según el chatbot con el que está interactuando, tendrán una ID, un mensaje, una pertenencia a un flujo y chatbot y palabras claves.
- **Usuario (User):** Serán los usuarios que serán registrados en el sistema, tendrán una ID y una lista el cual será el historial de interacción entre él y los chatbots.
- **Historial de chat (ChatHistory):** Corresponderá a la secuencia de palabras claves y números que el usuario utilizó en sus interacciones con los bots.

Paradigma: En este trabajo se dará el uso del “Paradigma Funcional”, el cual se basa en el cálculo lambda **(λ)** y en la composición de funciones puras evitando la modificación de las variables de entrada dentro de las mismas a la hora de implementar soluciones de software.

- **Composición de funciones:** En el contexto de algebra, consiste en una función compuesta de 2 o más funciones.
- **Funciones anónimas:** Derivada del cálculo lambda, son aquellas funciones los cuales se crean dentro de una función definida y se destruye luego de ser ejecutada, estos se representan mediante el símbolo lambda **(λ)**.
- **Funciones de orden superior:** Son aquellas funciones que toman una o más funciones como entrada, o devuelven una función como salida.

- **Currificación:** Transformación de una función de “n” argumentos de entrada, en una secuencia de “n” funciones de 1 argumento.
- **Recursividad:** Proceso en donde una función se llama así misma varias veces dependiendo del cumplimiento de ciertas condiciones, en este caso se implementaron la recursión natural y la recursión de cola.

Desarrollo

Análisis del problema:

Se solicitó la implementación de este sistema de chatbots a través del software de programación “DrRacket” el cuál usa el lenguaje funcional “ Scheme”, y esta implementación debe cumplir que se pueda crear chatbots, administrarlos e interactuar con estos. Para ello el elemento “Sistema” se usará como base para la creación y agregado de chatbots, flujos, opciones y usuarios. Sin embargo, para la interacción con algún chatbot se necesita que un usuario haya iniciado sesión en el sistema.

Las funciones requeridas (Anexo 1) para la implementación son las siguientes:

1. **Option:** Función constructora de una opción para flujo de un chatbot. Cada opción se enlaza a un chatbot y flujos especificados por sus respectivos códigos.
2. **Flow:** Función constructora de un flujo de un chatbot.
3. **Flow-add-option:** Función modificadora para añadir opciones a un flujo.
4. **Chatbot:** Función constructora de un chatbot.
5. **Chatbot-add-flow:** Función modificadora para añadir flujos a un chatbot.
6. **System:** Función constructora de un sistema de chatbots. Deja registro de la fecha de creación por make-system.
7. **System-add-chatbot:** Función modificadora para añadir chatbots a un sistema.
8. **System-add-user:** Función modificadora para añadir usuarios a un sistema.
9. **System-login:** Función que permite iniciar una sesión en el sistema.
10. **System-logout:** Función que permite cerrar una sesión abierta.
11. **System-talk-rec:** Función que permite interactuar con un chatbot.
12. **System-talk-norec:** Función que permite interactuar con un chatbot. Mismo propósito de la función anterior, pero con una implementación declarativa.
13. **System-synthesis:** Función que ofrece una síntesis del chatbot para un usuario particular partir de chatHistory contenido dentro del sistema.
14. **System-simulate:** Permite simular un diálogo entre dos chatbots del sistema.

Diseño de solución:

Con lo que se solicitó se implementó los siguientes TDA utilizando los elementos anteriormente mencionados:

- **TDA_system:** Lista de listas que contiene los usuarios registrados, el usuario que inició sesión y una lista de chatbots.
- **TDA_chatbot:** Lista de listas que contiene el nombre del chatbot en sí y una lista de flujos.
- **TDA_flow:** Lista de listas que contiene un mensaje entrante y una lista de opciones.
- **TDA_user:** Una lista que tendrá una ID y su historial de chat.
- **TDA_chatHistory:** Lista que contiene los elementos que ingresó el usuario mientras interactuaba con un chatbot.
- **TDA_option:** Lista de listas que contiene un mensaje y palabras clave.

La solución implementada consiste en la manipulación de datos del sistema y esto implicaba cualquier tipo de dato (números, strings, booleanos, etc.) utilizando las diferentes funciones distribuidas en los archivos TDA, y con este paradigma se permite un mejor manejo de dependencias teniendo menos puntos de cambios o fallos.

También de implemento un TDA_main en donde se definieron las funciones obligatorias que a su vez dependerá de todos los anteriores mencionados (Anexo 2)

Aspectos de implementación:

Para este laboratorio se usó como como compilador el mismo “DrRacket” versión 8.10, no se usó ninguna biblioteca externa al igual que tampoco se utilizó funciones “set!” ni derivados de este, ya que estas funciones emula un trabajo con variables y, en consecuencia, deja de utilizar el paradigma funcional.

En cada archivo TDA (a excepción del tda_main) se encuentran funciones que trabajan con los datos correspondientes a su campo (Anexo 3).

Instrucciones de uso:

Para empezar, se necesita que todos los TDAs estén guardados en una misma carpeta incluyendo el script de pruebas (en donde este se ejecutará el programa), si falta uno de estos el programa devolverá error donde se dirá que una función “X” no está definida.

Se puede definir un sistema vacío, pero para agregar un chatbot, se recomienda seguir el siguiente orden:

Crear opciones → Crear flujo con las opciones creadas → Añadir más opciones al flujo (opcional) → crear chatbot con los flujos creados → Añadir más flujos al chatbot (opcional) → Añadir el chatbot al sistema

Caso contrario el programa podría retornar solamente las opciones, los flujos o el chatbot.

Resultados y autoevaluación:

En el script de pruebas se probó con agregar 3 usuarios más junto con llamadas de login y logout, después se crearon 3 nuevos chatbots a los que a cada uno se le intentó agregar 1 flujo con 2 opciones. Al ejecutar el programa, las funciones implementadas (desde option hasta system-logout) dentro del plazo establecido entregan correctamente el sistema esperado, no se logró probar el interactuar con los chatbot debido a la falta de las funciones requeridas para ello siendo las funciones “system-talk-rec”, “system-talk-norec”, “system-synthesis” y “system-simulate”.

Por lo tanto, el programa implementado entrega de forma correcta el sistema de chatbots pero hasta el momento no hay forma de interactuar con ellos (Anexos 4).

Conclusión

Con lo obtenido anteriormente, se puede concluir que se ha logrado cumplir la mayor parte de lo solicitado para este laboratorio, el programa implementado retorna un sistema de chatbots de forma correcta, pero sin forma de interactuar con ella.

Durante el desarrollo del código fuente se han enfrentado dificultades al momento de manipular elementos a una lista ya sea sacar elementos o agregarlos a otra lista, siendo uno de estos problemas la presencia de listas dentro de otras listas de 1 elemento por lo que se tuvo que usar car para quitar los paréntesis innecesarios, otro fue que algunas funciones que tiempo después de ser creadas no terminaban de satisfacer lo esperado y en algunos caso se ha tenido que convertir algunas funciones en funciones de primer orden o simplemente se borraban y se creaban unas completamente nuevas.

Además, el uso del paradigma funcional a veces limitaba el cómo implementar algunas funciones generando en algunos casos una parálisis paradigmática.

Bibliografía

- Pablo Fernández (2022, septiembre 9) *Qué es la programación funcional y sus características.* Recuperado de https://openwebinars.net/blog/que-es-la-programacion-funcional-y-sus-caracteristicas/
- Cristian Tejedor (2020, abril) *Paradigmas de programación, programación funcional, función pura, lambda, list/dictionary comprehension.* Recuperado de https://www.infor.uva.es/~cvaca/asigs/docpar/paradigmas_sesion7.pdf
- Profesores de Paradigmas de programación (2023) *Repaso clase 5.* Recuperado de https://uvirtual.usach.cl/moodle/pluginfile.php/369148/mod_resource/content/1/Repaso%20Clase%205.pdf

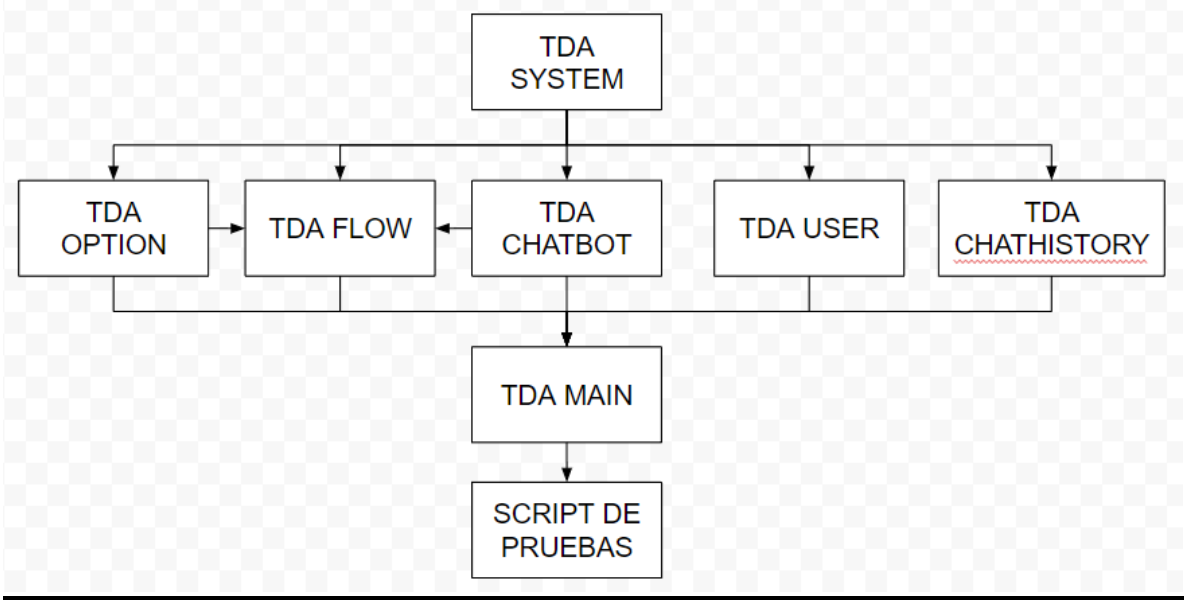
Anexos

1. Tabla de funciones obligatorias

Tipo de función	Nombre de función	Funcionamiento
Constructor	Option	Recibe los parámetros y mediante lamba, retorna una función dedicada a crear opciones, en el que este retorna una lista con su ID, un mensaje, la id del chatbot al que pertenece, la id del flujo al que pertenece y una lista con las palabras claves.
Constructor	Flow	Recibe los parámetros y mediante lamba, retorna una función dedicada a crear flujos, pero dentro de esa misma función se le aplica un filtro a las opciones de entrada donde se eliminan los duplicados comparando sus IDs.
Modificador	Flow-add-option	Esta función revisa si la opción de entrada ya está agregada con anterioridad al flujo comparando las IDs, si se cumple, retorna el flujo sin cambios, caso contrario, se agrega la opción al flujo.
Constructor	chatbot	Recibe los parámetros y mediante lamba, retorna una función dedicada a crear chatbots, en el que este retorna una lista con su ID, el nombre del chatbot, un mensaje de bienvenida, la id del flujo inicial al que pertenece y una lista de flujos al que se le aplicó el filtro para evitar duplicados.
Modificador	Chatbot-add-flow	Mediante recursión por cola, se revisa si el chatbot tiene flujos, si se cumple, se compara de uno en uno las ids de los flujos del chatbot con la del flujo de entrada, si se cumple, se retorna el chatbot sin cambios, caso contrario se agrega el flujo de entrada al chatbot,

Constructor	System	Recibe los parámetros y mediante lamba, retorna una función dedicada a crear sistemas, donde retorna una lista con la ID del sistema, el nombre del sistema, un espacio vacío para agregar los usuarios registrados, otro espacio en blanco para poner el usuario que inició sesión, la ID del chatbot inicial y una lista de chatbots al que se le aplicó el filtro de borrar duplicados, además de guardar la fecha y hora de su creación.
Modificador	System-add-chatbot	Esta función revisa si el sistema tiene chatbots, si se cumple, se comparan las id para verificar si el chatbot de entrada ya estaba agregado al sistema con anterioridad.
Modificador	System-add-user	Primero verifica si hay usuarios registrados en el sistema, si se cumple, revisa si el usuario de entrada ya fue registrado con anterioridad al sistema, para ello se compara la id del usuario con los demás. Caso contrario, retorna el sistema sin cambios.
Modificador	System-login	Primero verifica si hay usuarios registrados en el sistema, si se cumple, revista si un usuario anterior ya ha iniciado sesión, caso contrario, retorna el sistema de entrada sin cambios.
Modificador	System-logout	Empieza revisando si un usuario inició sesión, si se cumple se deja en blanco el espacio de la lista donde se deja el usuario conectado, caso contrario, retorna el sistema de entrada sin cambios
Las funciones obligatorias que no aparezcan en esta tabla, es porque no fueron definidas hasta ahora.		

2. Diagrama de dependencias entre los TDA implementados:



3. TDAs Implementados:

3.1. TDA-system:

Tipo	Función	Descripción
Constructor	Make-system	Se crea una lista con los elementos ingresados, además de dejar registro de la fecha de creación.
Selector	Get-system	Obtiene el nombre del sistema
Selector	Get-system-initialChatbot	Obtiene el id del chatbot inicial
Selector	Get-system-chatbots	Obtiene la lista de todos los chatbots agregados al sistema
Modificador	Filter-doubles-by-ID	Devuelve una lista sin elementos duplicados

3.2. TDA-option:

Tipo	Función	Descripción
Constructor	Make-option	Se crea una lista con los elementos ingresados, esta lista representará una opción
Selector	Get-option-id	Se obtiene la id de una opción

3.3. TDA-flow:

Tipo	Función	Descripción
Constructor	Make-flow	Se crea una lista con los elementos ingresados, esta lista representará un flujo
Selector	Get-flow-options-ids	Se obtiene todas las IDs de las opciones dentro de un flujo
Selector	Get-flow-id	Obtiene la id de un flujo
Selector	Get-flow-options	Obtiene las opciones de un flujo
Pertenencia	Flow-is-repeated	Verifica si el flujo ingresado ya existe dentro del chatbot ingresado
Modificador	chAddFlw-tail	Mediante una comparación de uno en uno, se verifica si el flujo de entrada ya existe dentro del chatbot.

3.4. TDA-user:

Tipo	Función	Descripción
Constructor	Make-user	Se crea una lista con el usuario y una lista vacía para su historial de chat
Pertenencia	Exists-user	Verifica la existencia de un usuario en el sistema
Pertenencia	A-user-conecter	Verifica si usuario ha iniciado sesión en el sistema
Selector	Get-user-id	Obtiene el id del usuario
Selector	Get-users-id	Obtiene una lista con las IDs de los usuarios registrados en el sistema
Modificador	Conect-user	Conecta el usuario de entrada para iniciar sesión
Selector	Get-users	Obtiene una lista de todos los usuarios registrados en el sistema
Selector	Get-user	Obtiene el nombre del usuario que ha iniciado sesión en el sistema
Modificador	Register	Registra el usuario ingresado al sistema

3.5. TDA-chatbot:

Tipo	Función	Descripción
Constructor	Make-chatbot	Se crea el un chatbot.
Selector	Get-chatbotID	Obtiene la ID de un chatbot.
Selector	Get-chatbot-flows	Obtiene los flujos de un chatbot.
Pertenencia	Is-chatbot-repeated	Verifica si el chatbot de entrada ya existe dentro del sistema.

Selector	Get-cahtbots-ids	Obtiene una lista de IDs de todos los chatbots del sistema.

3.6. TDA-chatHistory:

Tipo	Función	Descripción
Constructor	History	Crea una lista inicialmente vacía para almacenar el historial de chat de un usuario del sistema.

3.7. TDA-main: Anexo 1

4. Orden recomendado de las funciones para la creación de un chatbot para agregarlo al sistema:

Ejemplo de creación de chatbot

```
; nuevo chatbot (3)
(define op22 (option 1 "1) Videojuegos" 3 2 "juegos" "videojuegos"))
(define op23 (option 2 "2) Ropa" 3 1 "ropa" "prendas" "zapatos" "pantalones" "poleras"))
(define op24 (option 3 "3) Volver" 0 1 "volver" "salir" "regresar"))
(define f40 (flow 1 "Flujo 1 Chatbot3\n ¿Qué te gustaría comprar?" op22 op23 op24))
(define cb3 (chatbot 3 "Agencia de compras" "Bienvenido\n ¿Qué te gustaría comprar" 1 f40))
```

Salida ejemplo de chatbot

```
Welcome to DrRacket, version 8.10 [cs].
Language: racket, with debugging; memory limit: 128 MB.
> cb3
'(3
  "Agencia de compras"
  "Bienvenido\n ¿Qué te gustaría comprar"
  1
  ((1
    "Flujo 1 Chatbot3\n ¿Qué te gustaría comprar?"
    ((1 "1) Videojuegos" 3 2 ("juegos" "videojuegos"))
     (2 "2) Ropa" 3 1 ("ropa" "prendas" "zapatos" "pantalones" "poleras"))
     (3 "3) Volver" 0 1 ("volver" "salir" "regresar"))))))
>
```

5. Autoevaluación requerimientos no funcionales:

Requerimientos No Funcionales	Puntaje
Autoevaluación	1
Lenguaje	1
Verisón	1
Standard	1
No variables	1
Documentación	1
Dom->Rec	1
Organización	1
Historial	1
Script de pruebas	1
Prerrequisitos	0.75 No se definieron las funciones: system-talk-rec, system-talk-norec system-synthesis, system-simulate

6. Autoevaluación requerimientos funcionales:

Requerimientos Funcionales	Puntaje
TDA's	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	0 no se definió la función
system-talk-norec	0 no se definió la función
system-synthesis	0 no se definió la función
system-simulate	0 no se definió la función