UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Laboratorio Nº1 : Aplicación del Paradigma Funcional en un sistema simulador de chatbots

Nombre: Agustín Saavedra Olmos

Profesor: Gonzalo Martínez Ramírez

Asignatura: Paradigmas de Programación

09 de Octubre de 2023

ÍNDICE DEL INFORME

CAPÍTULO 1: INTRODUCCIÓN

En el presente informe Nº1 de la asignatura de Paradigmas de Programación, se aborda un proyecto acerca de un sistema simulador de chatbots interactivo, a través del paradigma funcional con el uso del lenguaje de programación Scheme. La estructura del informe contiene la introducción, descripción del problema y del paradigma, análisis del problema, diseño de la solución, aspectos de implementación, instrucciones de uso, resultados obtenidos, autoevaluación, conclusiones, referencias y finalmente anexos.

CAPÍTULO 2: DESCRIPCIÓN DEL PROBLEMA

La problemática de la experiencia consiste en la creación, despliegue y administración de un sistema de simulación de chatbots simplificado interactivo con el usuario, donde al sistema se le puede agregar chatbots, usuarios y chatHistory, a los chatbots se les puede agregar flujos y a los flujos se les puede añadir opciones con la implementación de operaciones como option, flow, flow-add-option, chatbot, chatbot-add-flow, etc. Después se procede a simular y mostrar el sistema de chatbots con la finalidad de interactuar con el usuario con funciones como system-talkrec, system-talknorec y system-synthesis.

CAPÍTULO 3: DESCRIPCIÓN DEL PARADIGMA

El paradigma funcional corresponde a un paradigma de tipo declarativo, el cual tiene como unidad fundamental de abstracción el uso de las funciones y tiene su orígen en el cálculo lambda. Se dice que es paradigma declarativo, ya que se especifica lo que debe hacer un programa (¿QUÉ?).

Los conceptos aplicados en este proyecto son las funciones anónimas, las funciones de orden superior y la recursión. Las funciones anónimas se expresan sin nombre, poseen una o más entradas y devuelven una salida. Las funciones de orden superior son funciones que en su entrada reciben una o más funciones y devuelven una función de salida. La recursión corresponde a una función que soluciona un problema a través de una llamada a sí misma. Existen tres tipos de recursión, recursión natural (forma básica, hay estados pendientes), recursión de cola (no hay estado pendiente,

se va construyendo progresivamente con una optimización de cola) y recursión arbórea (donde una función se llama múltiples veces a sí misma, formando la estructura de un árbol).

CAPÍTULO 4: ANÁLISIS DEL PROBLEMA

Para empezar a enfrentar el problema, se deben considerar los TDA's básicos mínimos con los que se pueden hacer los requerimientos funcionales del proyecto. En este caso, son 6 TDA's necesarios para hacer el sistema de chatbots son Opción, Flujo, Chatbot, Sistema, Usuario y ChatHistory.

El sistema de chatbots se encuentra compuesto por un nombre (string), código de enlace de chatbot inicial (entero), una lista de chatbots (lista), lista de usuarios (lista), usuario ingresado en el sistema (lista) y un chatHistory (lista), como a su vez los chatbots tienen un id (entero), nombre (string), mensaje de bienvenida (string), id de flujo inicial (entero) y una lista de flujos (lista). Los flujos por su parte, están formados por un id (entero), mensaje (string) y una lista de opciones (lista), así como las opciones se componen de un id (entero), mensaje (string), código de enlace a un chatbot (entero), código de enlace a un flujo inicial (entero) y una lista de palabras claves (lista), sin olvidar las otras entradas. En base a esto, se puede decir que la problemática anterior sigue una estructura de árboles (Anexo), debido a que el sistema tiene una lista de chatbots, los chatbots contienen una lista de flujos y los flujos contienen una lista de opciones. En este caso, la forma de resolver la problemática es utilizando las listas que corresponden a una de las estructura de datos más básicas del lenguaje de programación Scheme, ya que son más fáciles y cómodas de trabajar respetando el paradigma funcional que una estructura de tipo arbórea en este lenguaje.

Entonces, se confirma que para trabajar en el Sistema se necesitan los TDA's Opción, Flujo, Chatbot, Sistema, Usuario y ChatHistory para implementar funciones como system-add-user (en la cual se agrega un Usuario creando una lista nueva, considerando también los otras entradas del Sistema), system-logout (donde se elimina el usuario creando una lista con los elementos del Sistema, expresando lista vacía en el usuario registrado), etc. Finalmente, como el paradigma funcional es declarativo, se va creando una lista nueva para cada función de los TDA's.

CAPÍTULO 5: DISEÑO DE LA SOLUCIÓN

Por el análisis anterior se sabe que la estructura más importante de la problemática planteada es el TDA Sistema, debido a que contiene a todos los demás TDA's. Por otra parte, hay que considerar que la opción es la estructura más pequeña en el sistema de chatbots. Entonces, se debe crear la opción a través de listas con funciones como cons o list (cualquiera aplica), además se crean los selectores del TDA Opción, a través de car, cdr y sus variaciones. También se implementa el modificador que es una lista de opciones para la repetición de sus identificadores.

Después se puede implementar el TDA Flujo creando su constructor y sus selectores de la misma forma que el anterior. El constructor y el modificador del TDA Flujo verifica si el id de las opciones está repetido o no, a través de la funciones de orden superior map y filter, además que usa recursión natural para ver toda la lista de opciones.

En el TDA Chatbot se implementa el constructor verificando si los flujos contienen identificadores repetidos, sus selectores son en base a funciones de listas. Su modificador corresponde a chatbot-add-flow, este se implementa usando recursión natural a la repetición de identificadores de los flujos, en base a la función member.

Finalmente, el TDA Sistema sigue la mismas estructuras que las funciones anteriores, sin embargo sus modificadores system-add-chatbot y system-add-user verifican los identificadores repetidos de una lista de chatbots y de una lista de usuarios usando en la primera función recursión y en las otra la funcion member.

CAPÍTULO 6: ASPECTOS DE IMPLEMENTACIÓN

El proyecto contiene 7 tiene archivos que corresponden a cada una de las estructuras mínimas del problema, las cuales son "option", "flow", "chatbot", "system", "user" y "chatHistory". El otro archivo es el script de pruebas, donde se ejecutan los requerimientos funcionales y se muestra el funcionamiento del programa.

El proyecto se desarrolla en el compilador DrRacket versión 8.8, con el lenguaje de programación Scheme. No se usa ninguna biblioteca externa para respetar lo más posible el paradigma funcional y además facilitar el uso de funciones propias del lenguaje.

CAPÍTULO 7: INSTRUCCIONES DE USO

1) Los archivos de código de los TDA's y el script de pruebas deben estar en la misma carpeta de archivos para evitar posibles errores al momento de ejecutar el script de pruebas.

2) En la función flow-add-option, al ejecutar se ve que la opción agregada se encuentra primera que las opciones que están en el flujo correspondiente. Esto no significa ninguna alteración en el código, solamente se debe implementar de mejor forma. (Ver Anexo 1)

3) En la función system-add-chatbot se observa errores, debido a que no se añaden los chatbots de forma correspondiente al ejecutar esa parte del código. (Ver Anexo 2)

CAPÍTULO 8: RESULTADOS Y AUTOEVALUACIÓN

En cuanto a los resultados, se logra cumplir 9/11 requerimientos no funcionales de manera total (ver Anexo 3), por lo que se logra totalmente los resultados esperados en esta parte. Por otro lado, en los requerimientos funcionales se logra implementar 11/15 funciones, sin embargo no se pudo cumplir los resultados esperados (ver Anexo 4), debido a problemas en el análisis de abstracción de tales funciones y alguna de ellas no se llegaron a realizar. Las funciones faltantes son: system-talk-rec, system-talk-notrec, system-synthesis y system-simulate.

CAPÍTULO 9: CONCLUSIONES

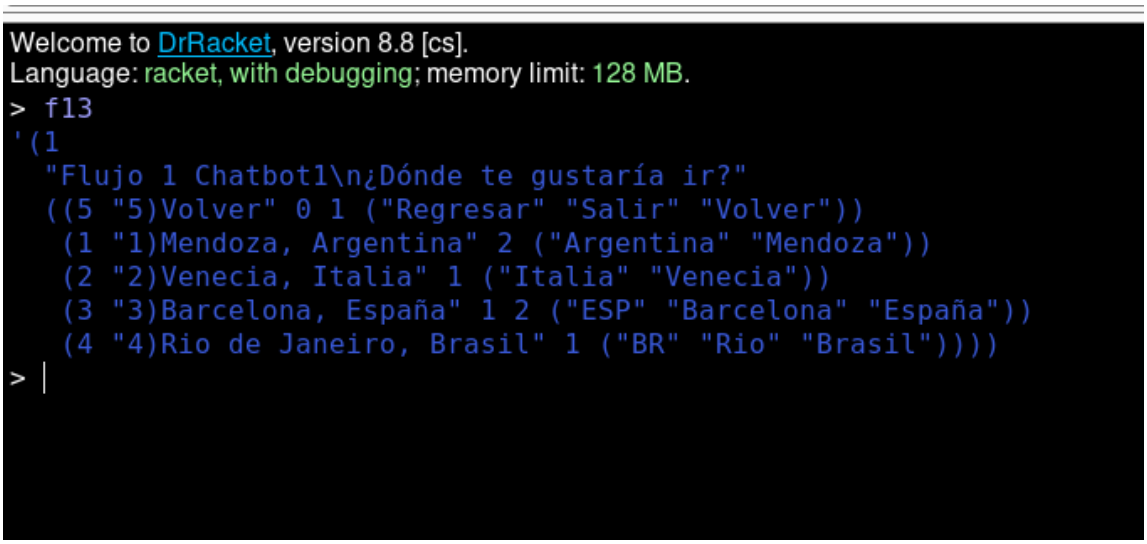
Con respecto al grado de alcance de los requerimientos, no se han logrado los objetivos propuestos del desarrollo de sistema de simulación de chatbots, debido a que hay fallas en la abstracción del problema planteado. Una de las limitaciones de este laboratorio es cambiar de paradigma, debido a que dejar de programar en un paradigma imperativo resulta ser complicado (parálisis paradigmática) y la otra limitación es la compleja abstracción del paradigma funcional que hace difícil aplicar algunos puntos en este laboratorio.

CAPÍTULO 10: REFERENCIAS

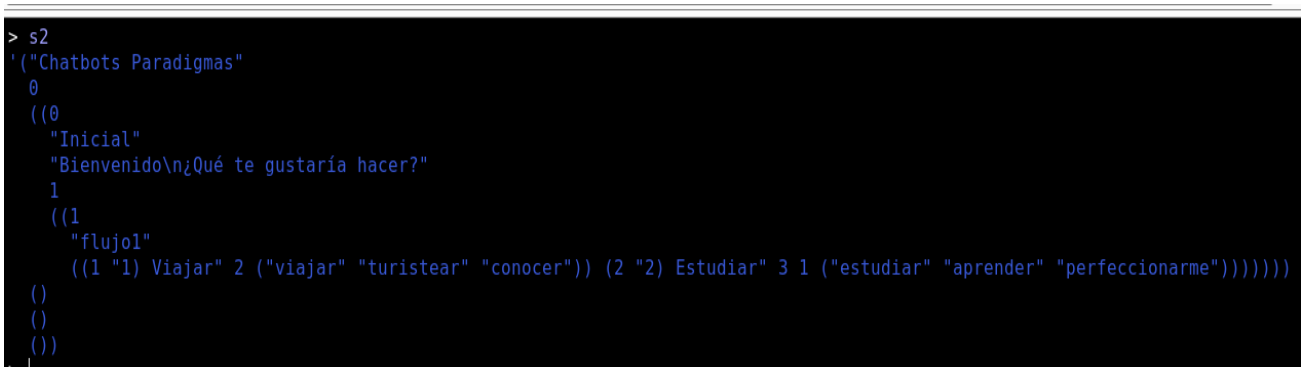
Rojas, A. (18 de noviembre de 2020). ¿Qué es la programación funcional?.*Incentro.*https://www.incentro.com/es-ES/blog/que-programacion-funcional

González, R. (2023). *3.- P.Funcional*. Paradigmas de Programación. UVirtual Usach. Recuperado de https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=11

CAPÍTULO 11: ANEXOS



Anexo 1: Ejecución de la función flow-add-option.



Anexo 2: Ejecución de la función system-add-chatbot

#	Requerimientos Funcionales	Puntaje
1	TDA's	0.75
2	option	1
3	flow	1
4	flow-add-option	1
5	chatbot	1
6	chatbot-add-flow	0.75
7	system	1
8	system-add-chatbot	0.5
9	system-add-user	1
10	system-login	1
11	system-logout	1
12	system-talk-rec	0
13	system-talk-norec	0
14	system-synthesis	0
15	system-simulate	0

Anexo 3: Autoevaluación de Requerimientos Funcionales.

#	Requerimiento No Funcional	Puntaje
1	Autoevaluación	1
2	Lenguaje	1
3	Versión	1
4	Standard	1
5	No variables	1
6	Documentación	1
7	Dom -> Rec	1
8	Organización	0.75
9	Historial	1
10	Script de Pruebas	1
11	Prerrequisitos	0.75

Anexo 4: Autoevaluación de Requerimientos No Funcionales