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

Informe de laboratorio 1 - Aplicación del paradigma funcional en el desarrollo de un sistema de chatbots

Alumno: Mohamed Al-Marzuk

Curso: Paradigmas de programación
Sección A-1
Profesor: Roberto González

9 de Octubre de 2023

Tabla de contenidos

1. Introduccion

Los paradigmas de la programación cuentan con un papel esencial para los desarrolladores, ofreciéndoles distintas maneras para la resolución de problemas y en la creación de software. En esta primera entrega de laboratorio, se aplicarán los conceptos del paradigma funcional en el desarrollo de un chatbot, utilizando el lenguaje de programación "Scheme", a través del interprete "DrRacket", en su versión 8.10. (Gonzalez (2023b)).

1.1. Descripción del Problema

Como se mencionó antes, el propósito de esta entrega es crear un sistema de ITR (Respuesta de Interacción a Texto) con varios chatbots que, en base a la palabra clave que se le entregue a alguno de ellos, se derive la atención del usuario al chatbot que se relacione con lo que este busca. Estas palabras se encuentran almacenadas con anterioridad en la estructura del chatbot, por lo que son lo único que este puede entender. Este tipo de chatbots no cuentan con inteligencia artificial.

1.2. ¿Qué es el paradigma funcional?

El paradigma funcional es un paradigma declarativo. El principal enfoque de este paradigma está en la utilización de funciones, que se caracterizan por no admitir la manipulación de variables, pues estas no existen en este paradigma. También se destaca por la ausencia de ciclos y condicionales. Como en matemáticas, cada función posee un dominio y un recorrido, y estas se caracterizan por aceptar tanto tipos de datos, como también otras funciones. (Navarro (1989)). Para el desarrollo de esta primera entrega de laboratorio, se utilizarán los siguientes conceptos propios de la programación funcional:

1. **Recursión:** Se define la función en terminos de sí misma. Cuenta con un caso base y un caso recursivo, en donde se hace el llamado a la misma función. El caso base cuenta además con una condición de borde y una solución conocida.

2. **Funciones anónimas**: Es un tipo de función sin nombre que, en el momento en el que son definidas, son evaluadas. Se utiliza el cálculo lambda para la creación de estas.

2. Desarrollo

2.1. Análisis del problema

Para el desarrollo de este sistema de chatbots, se identifican como TDA los siguientes elementos fundamentales que deben ser implementados:

1. **Sistema:** El sistema se representa como una lista que tenga, en el siguiente orden, los siguientes elementos: nombre del sistema (string), ID del chatbot inicial (int), lista de usuarios registrados (list), lista con el usuario actual (list), lista con el chat history, lista con todos los chatbots del sistema (list). **(Tabla 1).**

2. **Chatbot:** El chatbot está representado en forma de lista con elementos que lo representen. La lista contiene el ID del chatbot (int), su nombre (string), su mensaje de bienvenida (string), el ID del flujo inicial (int) y una lista de flujos (list). **(Tabla 2).**

3. **Flujo:** El flujo es una lista que contiene los elementos que lo representan. Estos son el ID del flujo, su nombre y una lista de opciones (list). **(Tabla 3).**

4. **Opción:** La opción es una lista que contiene el ID de la opción (int), su nombre (string), el ID del chatbot al que se asocia (int), el ID del flujo al que se asocia (int) y una lista con una o más palabras claves asociadas (list). **(Tabla 4).**

5. **Chat History:** El chat history está conformado por una lista con el nombre del usuario como primer elemento (string), seguido por todos los mensajes que este ha mandado. Mientras que los mensajes son una lista compuesta por el tiempo actual (int), el usuario que lo envía (string) y el mensaje en sí (string). **(Tabla 5).**

6. **Usuario:** El usuario es simplemente un string.

Además, cada uno de estos TDA debe incluir todas las funciones requeridas para llevar a cabo las funcionalidades específicas solicitadas en el laboratorio. Las funciones constructoras, selectoras, modificadoras y de pertenencia de cada TDA se encuentran en las tablas mencionadas.

2.2. Diseño de la solución

El diseño de solución, utilizando la programación funcional, consiste en la creación y desarrollo de distintas funciones para cumplir con lo requerido por el programa. Primero, se implementan los TDA que se mencionaron anteriormente, con las funciones necesarias, a la vez que suficientes, para que estos puedan trabajar correctamente. El TDA principal es el Sistema, el cual contiene a todos los demás. Ahora, ¿Cómo está construido el programa? El sistema incluye diversa información útil para el correcto funcionamiento del programa, entre ellas, se encuentra el chat history, usuarios registrados y usuarios logeados, teniendo una relación entre estos tres elementos. Por ejemplo, si el usuario no se encuentra registrado, no podrá iniciar su sesión en el sistema, la función system-login se encarga se hacer esa comprobación, por otro lado, si no está logeado, no podrá registrar sus mensajes en el chat history, las funciones encargadas de hacer esa comprobación son system-talk-rec y system-talk-norec **(Figura 1)**. Dentro del sistema también existe la lista de todos los chatbots que forman parte de él, teniendo un chatbot inicial, que se relaciona con el resto de chatbots del sistema a través de los flujos. Estos últimos tienen asociadas las opciones que tiene el usuario para elegir. En resumen, un sistema contiene un grupo de chatbots, un chatbot contiene una serie de flujos, y un flujo contiene una serie de opciones **(Figura 2)**.

2.3. Aspectos de implementación

Para el desarrollo del proyecto, se utilizó Dr. Racket versión 8.1 en su versión estándar, es decir, no se importaron bibliotecas externas, y se utilizaron solo las funciones base de Racket. Esto es debido a que se debe trabajar con el paradigma funcional. Es por esto que tampoco se utiliza la función "set". Gonzalez (2023a)

2.4. Instrucciones de uso

Para poder ejecutar el programa, se debe crear una carpeta en donde se encuentren los archivos de cada uno de los TDA implementados para solucionar el problema planteado. Después, se debe crear un archivo de prueba en la misma carpeta, el cual tiene que importar todos los archivos de los TDA para poder hacer uso de sus funciones.

1. **Ejemplo de uso**: Dentro del archivo de prueba, tendremos, por ejemplo, la función constructora de opciones, que recibe los cinco parámetros que fueron mencionados anteriormente y crea una lista que los contenga. Es importante que se respete el mismo orden que se especifica, para que no haya errores por usos incorrectos de la función. Otro ejemplo es la función flow-add-option, que recibe como argumentos un flujo y una opción, y agrega la opción a un flujo, y se encarga de filtrar las opciones en base a su ID.

2. **Resultados esperados:** Se espera que el sistema de chatbots esté implementado correctamente en un 80 %, pues casi todas las funcionalidades realizan su cometido de forma precisa y sin presentar errores. Además, se espera que cada función creada sea utilizada sin problemas en el script de pruebas.

3. **Posibles errores:** Las funciones system-talk-rec y system-talk-norec no están implementadas al 100 %, por lo que pueden presentar errores. Estas dos funciones no sirven si se les entrega un número, solo sirven si se les entrega una palabra clave.

2.5. Resultados

Las funciones implementadas, desde el RF2 hasta el RF11, funcionaron perfectamente. Las funciones constructoras se encargan de eliminar los elementos duplicados que se les pasen como argumento, y las funciones modificadoras se encargan de comprobar si el elemento a agregar es válido o no. Con respecto al RF12 y RF13, estos no fueron implementados correctamente. y el RF14 realiza correctamente su cometido al mostrar el historial del usuario solicitado. Por último, el RF15 no fue implementado por agotar el tiempo en las funcionalidades anteriores.

2.6. Autoevaluación

Los RF funcionan correctamente, menos el RF12 y RF13. En el caso del primero, está implementado en un 75 %, pues agrega al historial las interacciones entre el chatbot y el usuario, sin embargo, no sigue el sistema de flujos que debería seguir. Con respecto al

RF13, se implementó en un 25 %, pues lo único que hace es agregar el mensaje del usuario al historial, faltando la respuesta del chatbot y seguir el sistema de flujos. El RF14 muestra el historial del usuario que se le pide, así que está al 100 %. El RF15 no fue implementado, así que está al 0 %. En el archivo de autoevaluación hay más detalles al respecto.

3. Conclusión

Tras haber implementado la mayoría de los requerimientos funcionales solicitados en el enunciado de la entrega, se puede concluir que se aplicó de forma correcta, o casi correcta el paradigma funcional para el diseño de un sistema de chatbots. La principal complicación que se obtuvo durante el desarrollo de esta entrega viene ligada al desarrollo del RF12 system-talk-rec, esto es debido a que no se logró implementar correctamente esta función, y eso fue agotando poco a poco el tiempo que quedaba para el término del plazo. Esto provocó que no hubiese tiempo suficiente para poder pensar en el resto de requerimientos funcionales, y esto ocasionó que la implementación final del RF13 tampoco fuese correcta. Con respecto a las dificultades para utilizar la programación funcional, se destaca la ausencia de variables y estructuras de control de flujo, como los ciclos. Estos elementos, que son más comunes en la programación imperativa, habrían sido de suma utilidad para el desarrollo de esta actividad. En definitiva, programar bajo este paradigma es muy distinto a la programación en lenguajes como Python o C.

Finalmente, se puede concluir que, al programar bajo estas "nuevas reglas", la mente se amplía hacia nuevos horizontes y se adquieren nuevos conocimientos de programación, los cuales serán sumamente útiles para el desarrollo de futuros proyectos, ya sea en la universidad o en la vida profesional.

Bibliografía

Gonzalez, R. (2023a). Paradigma funcional. https://docs.google.com/document/d/1lErqBgtZSLxtedB4dsQjmg6ujSi5GwfJ-NKjsRvlXKY/edit.

Gonzalez, R. (2023b). Paradigmas de programación, proyecto semestral de laboratorio. https://docs.google.com/document/d/1L-B2b3J71Baqa_IuZt6EmRwDlxC0qzWn9YJAm6FIiJk/edit.

Navarro, J. J. M. (1989). Diseño, semántica e implementación de babel: Un lenguaje que integra la programación funcional y lógica. https://dialnet.unirioja.es/servlet/articulo?codigo=4902462.

Anexos

Tabla 1 - Funciones del TDA System		
Tipo de función	Nombre	Descripción
Constructor	system	Construye un sistema
Pertenencia	system?	Comprueba si un dato es system
Selector	system-get-chat-history	Entrega el chat history
Selector	system-get-logged-user	Entrega el usuario logeado
Selector	system-get-chatbots	Entrega los chatbots
Selector	system-get-register-user	Entrega los usuarios registrados
Modificador	system-add-chatbot	Añade un chatbot al sistema
Modificador	system-add-user	Registra un usuario en el sistema
Modificador	system-login	Logea a un usuario al sistema
Modificador	system-logout	Deslogea a un usuario del sistema
Selector	system-get-key-list	Entrega todas las keywords
Selector	system-get-key-list2	Entrega una lista de keyword específica
Pertenencia	msg-is-keyword?	Comprueba si un mensaje es keyword
Selector	system-specific-chatbot	Accede a las opciones según la keyword
Modificador	system-talk-rec	Habla con el chatbot recursivamente
Modificador	system-talk-norec	Habla con el chatbot no recursivamente
Selector	system-synthesis	Entrega una síntesis del historial de un usuario
Otros	modificador	Modifica el elemento de un indice de la lista

Tabla 2 - Funciones del TDA Chatbot		
Tipo de función	Nombre	Descripción
Constructor	chatbot	Construye un chatbot
Pertenencia	chatbot?	Comprueba si un dato es chatbot
Selector	chatbot-get-id	Entrega el id del chatbot
Selector	chatbot-get-msg	Entrega el msg de bienvenida del chatbot
Selector	chatbot-get-name	Entrega el nombre del chatbot
Selector	chatbot-get-flows	Entrega la lista de flujos del chatbot
Modificador	chatbot-add-flow	Agrega un flujo al chatbot.
Selector	chatbot-get-key-list	Entrega la lista de keywords del chatbot
Selector	chatbot-specific-flow	Entrega las opciones en base a una keyword

Tabla 3 - Funciones del TDA Flow		
Tipo de función	Nombre	Descripción
Constructor	flow	Construye un flujo
Pertenencia	flow?	Comprueba si un dato es flujo
Selector	flow-get-options	Entrega las opciones del flujo
Modificador	flow-add-option	Añade una opción al flujo
Pertenencia	id-repetido?	Comprueba si el id de un flujo a añadir se repite
Selector	flow-get-name	Entrega el nombre del flujo
Selector	flow-specific-option	Entrega las opciones en base a una keyword

Tabla 4 - Funciones del TDA Option		
Tipo de función	Nombre	Descripción
Constructor	option	Construye un flujo
Pertenencia	option?	Comprueba si un dato es flujo
Selector	option-get-keywords	Entrega las keywords de la opción
Selector	option-get-id	Entrega el ID de la opción
Selector	option-get-option	Entrega el nombre de la opcion

Tabla 5 - Funciones del TDA Chat-History		
Tipo de función	Nombre	Descripción
Constructor	chat-msg	Construye un mensaje de chat-history
Constructor	chat-history	Construye un chat-history de usuario
Pertenencia	chat-msg?	Comprueba si un dato es mensaje
Modificador	add-chat-history	Añade un mensaje a un chat-history

Figura 1: Diagrama ChatHistory

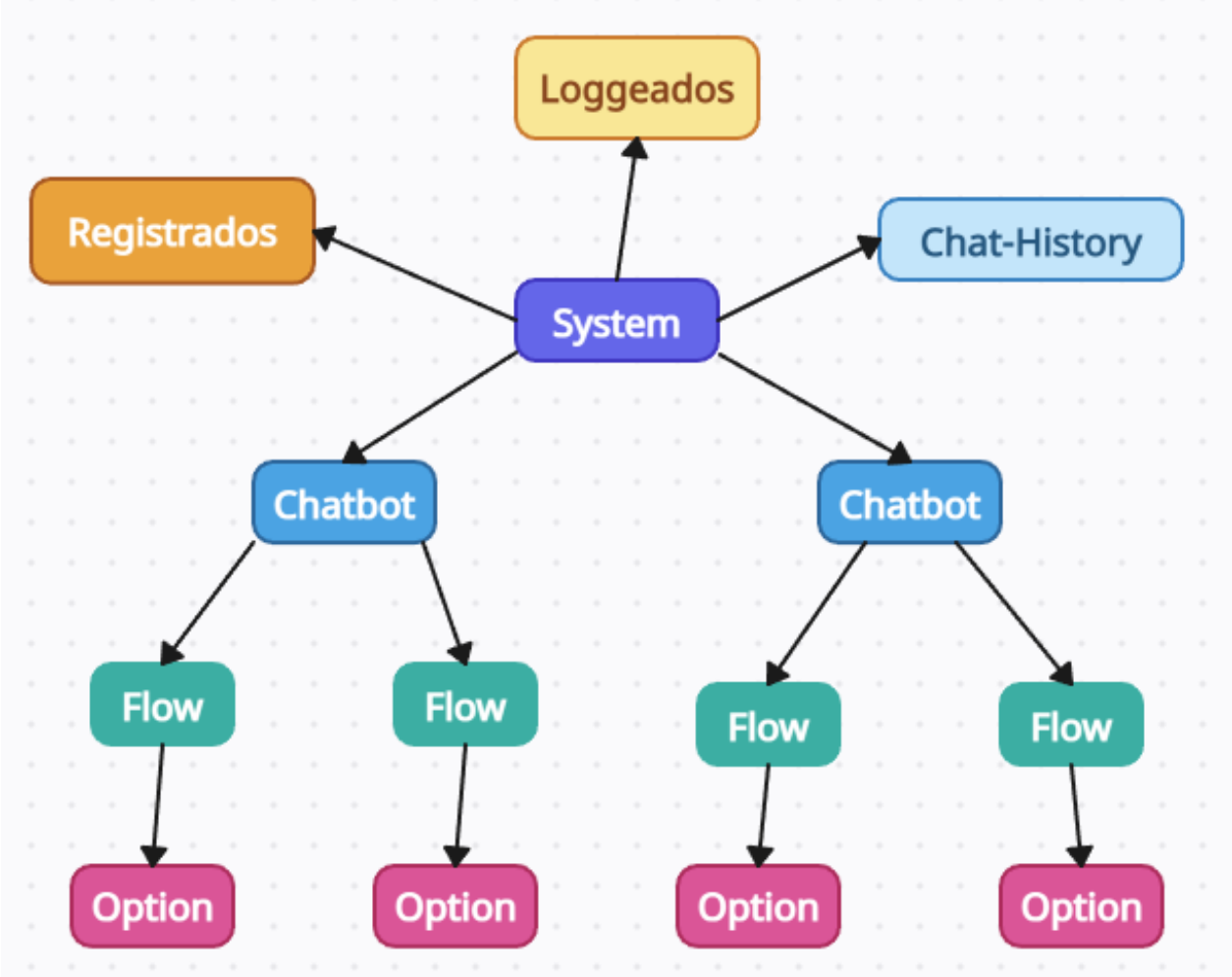

Figura 2: Partes de un sistema