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

Informe de laboratorio 2 - Aplicación de la programación lógica en el desarrollo de un sistema de chatbots

Alumno: Mohamed Al-Marzuk
Curso: Paradigmas de programación
Sección A-1
Profesor: Roberto González

Contenido

Introducción 3

Descripción del problema 3

¿Qué es la programación lógica? 3

Desarrollo 4

Análisis del problema 4

Diseño de la solución 5

Aspectos de implementación 5

Instrucciones de uso 5

Resultados 6

Autoevaluación 6

Conclusión 7

Bibliografía 8

Anexos 9

Introducción

El propósito de este informe es presentar el diseño de un simulador de un sistema de chatbots, mostrando la interacción entre el usuario y las distintas opciones que contiene cada flujo de un chatbot. Para esto, se utilizarán los conceptos del paradigma de la programación lógica, y se implementará la solución en el lenguaje de programación Prolog, utilizando el intérprete ”SWI-Prolog”, en su versión 9.1. A lo largo de este documento se analizarán los antecedentes del proyecto, al igual que las motivaciones que llevan a realizarlo, y se expondrán los detalles del proceso de solución del problema, utilizando la programación lógica. Se procederá a realizar primero una descripción del problema a abordar, y del paradigma en sí. tras esto, se verán aspectos importantes, como el diseño de la solución y las instrucciones de uso. Finalmente, se realizará un análisis de los resultados, junto a una autoevaluación acerca del trabajo realizado.

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, o del número 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. Lo que se busca es crear un sistema que contenga a varios chatbots, cada uno de ellos está asociado a una actividad en concreto.

¿Qué es la programación lógica?

El paradigma de programación lógica es un paradigma declarativo. Este paradigma opera gracias a la base de conocimientos que se le brinda al declarar ***hechos***, que el programa asumirá como verdades cada vez que sean consultados, y ***reglas***, que permiten deducir nuevos conocimientos a partir del que ya posee. Dentro de este paradigma, existen diversos elementos a tener en cuenta, por lo que se procederá a mencionar algunos. En primer lugar, tenemos la **unificación**, esta se refiere a hallar el término que haga que una consulta entregue verdadero. Luego, tenemos la ***consulta*** que es una pregunta que se le hace a la base de conocimientos. El programa encuentra respuesta en base a lo que sabe, pudiendo retornar verdadero si es que se le entrega un hecho, o pudiendo unificar una variable para que su resultado sea verdadero. Si no encuentra respuesta, entregará falso. Los ***predicados*** son expresiones lógicas que establecen una relación entre uno o más términos. Pueden ser hechos o reglas. Una regla puede ser ***recursiva***. Para esto, hay que declarar un hecho que represente una condición de parada, y tras esto, se define una o más reglas en términos de sí mismas, de forma que se pueda llegar a la condición de parada.

Desarrollo

Análisis del problema

Antes que nada, para poder realizar un correcto análisis del problema es necesario identificar correctamente los TDAs fundamentales para poder realizar este simulador de un sistema de chatbots. Estos son: el sistema que contiene a los chatbots, el chatbot en sí, los flujos de un chatbot y las opciones de un flujo. Además de estos cuatro, es necesario tener una buena representación de un usuario, además de contar con una buena construcción del chat-history. A continuación, se brindará más detalle sobre estos TDAs.

- **Usuario**: Es simplemente un string, los usuarios pueden interactuar con los chatbots si es que están loggeados, y tienen un chat-history en el sistema.
- **Chat-History:** El chat-history de un usuario está construido como una lista dentro del sistema. Y esa lista tiene como primer elemento al nombre del usuario, y como resto de elementos, tiene los mensajes tanto del usuario como del chatbot. Los mensajes también están construidos como una lista, teniendo el tiempo de envió del mensaje, el emisor, y el mensaje en sí. Además de los espacios y de los saltos de línea que agrega el constructor del mensaje **(Tabla 1)**.
- **Opción**: Es una lista que contiene el ID de la opción, su nombre, el ID del chatbot al que se asocia, el ID del flujo al que se asocia y una lista con una o más palabras claves asociadas **(Tabla 2)**.
- **Flujo**: Es una lista que contiene su ID, su nombre, y una lista que contiene a varias opciones **(Tabla 3)**.
- **Chatbot:** Este se representa como una lista que contiene su ID, su nombre y su mensaje de bienvenida, el ID del flujo actual, y una lista con varios flujos **(Tabla 4)**.
- **Sistema:** El sistema es una lista que contiene su nombre, el ID del chatbot actual, una lista con los usuarios registrados, un string que representa al usuario loggeado, en el caso de que no haya ninguno, es una lista vacía, una lista con el chathistory de todos los usuarios, y una lista con todos los chatbots del sistema **(Tabla 5)**.

Teniendo esto en cuenta, queda hablar de cómo se relacionan estos elementos, con respecto a la interacción, esta se basa en uno o más flujos estructurados dentro de un chatbot, que contienen opciones. Las palabras claves o los IDs de las opciones son las que facilitan esta interacción. El sistema puede gestionar uno o más chatbots, y también guarda la información de uno o de varios usuarios. El chat-history permite representar de manera clara la conversación de un usuario. Ahora, para poder hacer una buena implementación de estos TDAs, se deben definir correctamente las operaciones y las funcionalidades necesarias y suficientes para asegurar una correcta relación entre ellos.

Diseño de la solución

El diseño de la solución, utilizando el paradigma de programación lógica, consiste en la creación de distintas reglas y predicados para cumplir con lo requerido por el problema. Primero, se tiene que hacer la implementación de los TDAs mencionados en el análisis del problema, junto a las reglas esenciales para que estos funcionen correctamente. Podemos ver que el TDA principal es el sistema, pues es el que contiene a todos los demás. El sistema incluye diversa información útil para el correcto funcionamiento del programa, tales como la lista de usuarios registrados, el usuario loggeado y el chat-history de los usuarios. Existe una relación entre estos tres elementos. Por ejemplo, si el usuario no se encuentra loggeado en el sistema, no podrá agregar sus mensajes al chat-history, de hecho, tampoco podrá interactuar con el sistema de chatbots, pues Prolog arrojará falso si no encuentra un usuario, la regla que hace esta comprobación es systemTalkRec/3. Además, si el usuario no está registrado, no podrá iniciar sesión, esta comprobación la hace systemLogin/3, al igual que hace la comprobación de si hay un usuario loggeado en el sistema, en el caso que lo haya, no se podrá iniciar sesión, y Prolog arrojará falso, la ***figura 1*** muestra la relación entre estos elementos. Dentro del sistema, también está la lista de todos los chatbots. El sistema tiene un id de chatbot inicial, el cual será el primero en aparecer frente al usuario, cuando este le diga “hola”. Ese chatbot inicial mostrará las opciones que se encuentran contenidas en su flujo inicial. Cuando el usuario hace la interacción con systemTalkRec/3, se accede al flujo actual del chatbot actual y se busca la keyword o la opción dentro de ese flujo, cuando la encuentre, se recogen las IDs del chatbot y flujo asociados a la opción, se actualiza el ID del flujo dentro del chatbot, y el ID del chatbot dentro del sistema. De esta forma, se puede asegurar una correcta interacción entre el usuario y el sistema. La ***figura 2*** muestra cómo está construido el sistema.

Aspectos de implementación

Para la implementación de este problema, se utilizó SWI-Prolog 9.1 en su versión estándar, es decir, no se importó ninguna librería externa al lenguaje, y se utilizaron únicamente los predicados de base de Prolog, además de los predicados que fueron definidos por mí mismo.

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. El código fuente tiene 6 archivos, siendo estos tdaoption_22594262_Al-Marzuk.pl, tdaflow_22594262_Al-Marzuk.pl, tdachatbot_22594262_Al-Marzuk.pl, tdasystem_22594262_Al-Marzuk.pl, tdachathistory_22594262_Al-Marzuk.pl y main_22594262_Al-Marzuk.pl. Cada uno de estos archivos contiene las reglas y hechos de cada uno de los TDA, con sus constructores, pertenencia modificadores y selectores, mientras que el archivo main contiene los RF obligatorios del enunciado. Teniendo esto, se debe abrir SWI-Prolog y consultar al archivo main, esto se hace copiando los contenidos de los archivos prueba1_22594262_Al-Marzuk.pl y

prueba2_22594262_Al-Marzuk.pl y pegándolos en el apartado de consultas. El resultado de la interacción con el chatbot se mostrará por consola.

- **Ejemplos de uso**: Dentro de los archivos de pruebas, tendremos por ejemplo la siguiente consulta al predicado option/6: option(1, "1 - viajar", 1, 1, ["viajar", "turistear", "conocer"], OP1). Esta consulta unificará OP1 como una lista con los cinco elementos anteriores. Es importante mantener el orden especificado dentro del dominio en la consulta, pues en caso contrario, podrían haber errores, como que entregue falso la consulta. Otro ejemplo a mencionar es la siguiente consulta al predicado systemAddUser/3: systemAddUser(S02, "user1", S2), esta consulta toma el sistema S02, el string “user1”, que representa un usuario, y lo unifica en S2, que es un sistema con “user1” entre los usuarios registrados. Para esto, hay que tener construido el sistema S02 con el predicado system/4. También hay que tener en cuenta que “user1” no tiene que estar registrado en el sistema, de caso contrario, esta consulta arrojará falso.
- **Resultados esperados**: Se espera que el sistema de chatbots esté implementado correctamente en un 90%, pues todos los RF cumplen su cometido de manera precisa sin mostrar errores. Se espera que las consultas al main no presenten ningún inconveniente y que los scripts funcionen de manera correcta.
- **Posibles errores**: Se esperan errores en el predicado systemSynthesis/3, pues en el caso de que se le pida síntesis de un usuario no registrado, debería arrojar falso, siguiendo la lógica de los predicados anteriores. Sin embargo, en mi implementación, arroja un mensaje de “Usuario no encontrado”, esto es debido a que el programa se caía si no agregaba ese caso.

Resultados

Todos los requerimientos funcionales, desde el RF1 hasta el RF13, fueron implementados correctamente. Los constructores funcionan correctamente, pues eliminan los elementos duplicados en caso de que los haya, mientras que los modificadores se encargan de comprobar si los elementos a agregar son válidos o no. Esto se puede ver en el script de pruebas. En el caso de que algún elemento no sea válido, la consulta arrojará falso. El RF14 no fue implementado.

Autoevaluación

Los RF funcionan correctamente. No hubo ningún problema en su implementación. Con respecto al systemSynthesis/3, el único error fue el que se mencionó con anterioridad, sin embargo, con el caso extra que fue agregado, ese error ya fue corregido. El Rf14, al no ser implementado, está al 0%. En el archivo de autoevaluación hay más detalles al respecto.

Conclusión

Tras haber implementado la mayoría de los requerimientos funcionales del laboratorio, se puede concluir que se aplicaron correctamente los principios de la programación lógica para el desarrollo de un simulador de un sistema de chatbots. La principal complicación que se tuvo para desarrollar esta entrega viene ligada al RF12, pues fue el requisito funcional que más tiempo de desarrollo requirió, además de que fue el que más líneas de código necesitó para funcionar correctamente. Además, fue una funcionalidad que tampoco pude implementar al 100% en la primera entrega, por lo que tampoco tenía una idea clara de cómo realizarla. Sin embargo, para esta ocasión, sí que logré implementarla en su totalidad. Con respecto a las dificultades de utilizar la programación lógica, se destaca la ausencia de variables y ciclos, que son elementos más comunes en la programación imperativa, habrían sido de mucha ayuda en caso de tenerlos. Sin embargo, si se compara la programación lógica con la programación funcional, se puede afirmar que el paradigma de esta entrega es mucho más intuitivo a la hora de trabajar, pues en la primera entrega, era realmente un mareo ver tantos paréntesis, y era mucho más complicado ver a simple vista lo que hacía una función, pues podríamos estar frente a una composición de varias funciones. En la programación lógica, en cambio, teniendo la unificación, se puede entender de una mejor forma un bloque de código, pues las variables unificadas se pueden utilizar en otra consulta, sin necesidad de escribir el predicado dentro de la misma. En resumen, fue menos tortuoso trabajar con la programación lógica en Prolog que con la programación funcional en Racket.

En conclusión, se logró adquirir conceptos importantes sobre la programación lógica y sobre el lenguaje Prolog, y lo más importante, se pudo ampliar la mente hacia nuevos conocimientos al estar sujetos a programar bajo estas “nuevas reglas”. No cabe duda de que los conocimientos adquiridos serán de suma utilidad tanto para la vida laboral como para el resto de la vida universitaria.

Bibliografía

Gonzalez, R. (2023, septiembre 30). *Proyecto semestral paradigmas*. https://docs.google.com/document/d/1L-B2b3J71Baqa_IuZt6EmRwDlxC0qzWn9YJAm6FIiJk/edit

Gonzalez, R. (2023, octubre 10). *Paradigma de la programación lógica*. https://docs.google.com/document/d/1QCk3k-HCShNSByEZHoEIFbrxoOJSnPblyjiJDj3Q9Jk/edit

Castro, K. A., Juan, G., & Rojas Sanchez, D. (s/f). *PROGRAMACIÓN LÓGICA*. Github.io. Recuperado el 13 de noviembre de 2023, de https://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/logica_teoria/docs/2017-1.pdf

Anexos

Figura 2 – Representación Chat-History

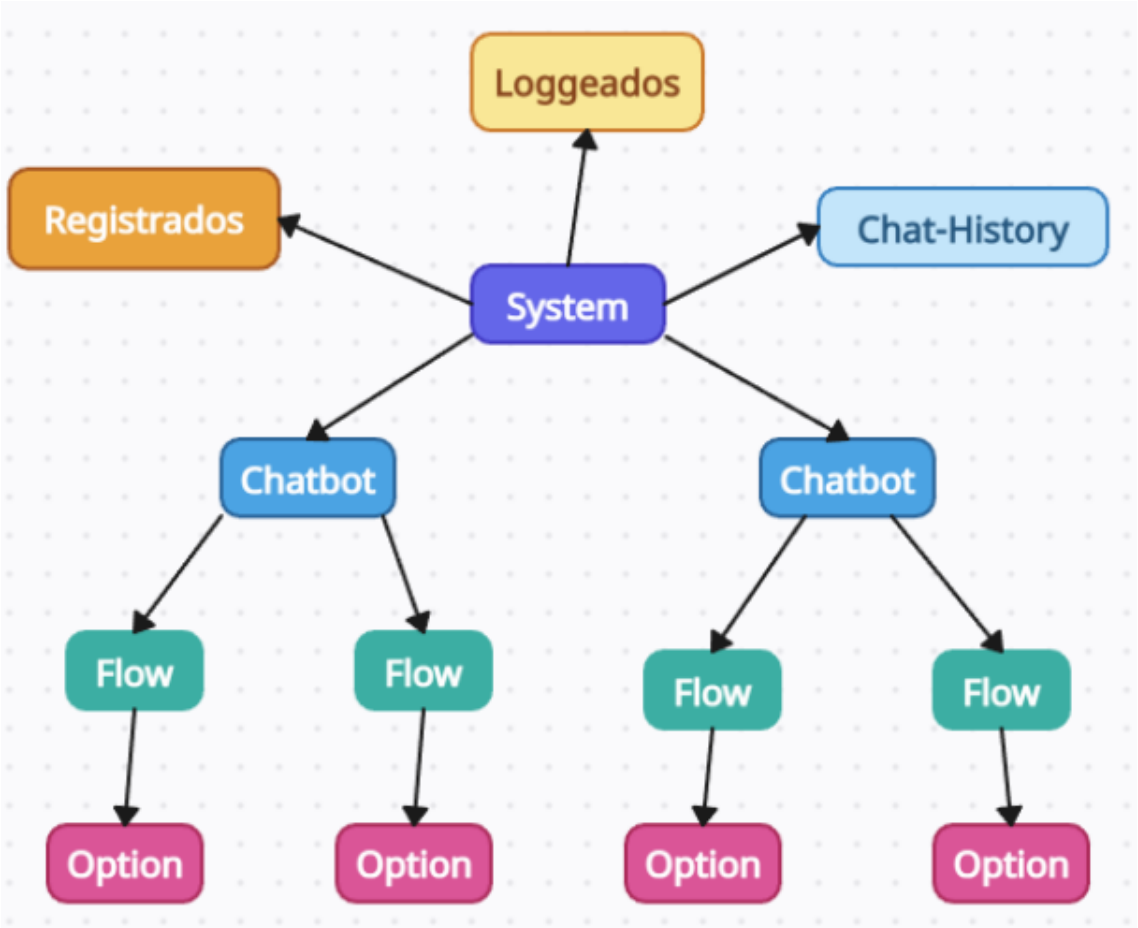

Figura 2 – Organización de un sistema

Tabla 1 - Predicados del TDA ChatHistory		
Tipo de predicado	Nombre	Descripción
Constructor	createUserHistory/4	Construye un mensaje de usuario
Constructor	createChatbotHistory/4	Construye un mensaje de chatbot
Modificador	chatMsgConcat/2	Concatena en un string un mensaje
Modificador	chatHistoryConcat/2	Concatena en un string un chathistory

Tabla 2 - Predicados del TDA Option		
Tipo de predicado	Nombre	Descripción
Constructor	option/6	Construye una opción
Selector	optionGetChatbotID/2	Entrega el ID del chatbot
Selector	optionGetKeywords/2	Entrega las keywords de la opción
Selector	optionGetID/2	Entrega el ID de la opción
Selector	optionGetName/2	Entrega el nombre de la opcion
Selector	optionGetFlowID/2	Entrega el ID del flujo
Selector	optionIdsFilter/3	Entrega una lista con el ID del chatbot y flujo
Selector	optionIdsFilterNum/3	Entrega una lista con el ID del chatbot y flujo
Otros	isNull/1	Comprueba si un elemento es una lista vacía

Tabla 3 - Funciones del TDA Flow		
Tipo de predicado	Nombre	Descripción
Constructor	flow/4	Construye un flujo
Modificador	flowAddOption/3	Añade una opción al flujo
Selector	flowGetID/2	Entrega el ID del flow
Selector	flowGetOptions/2	Entrega las opciones del flujo
Selector	flowGetName/2	Entrega el nombre del flujo
Selector	flowIdsFilter/3	Entrega una lista con el ID del chatbot y flujo
Selector	flowIdsFilterNum/3	Entrega una lista con el ID del chatbot y flujo
Selector	flowOptionSearch/2	Entrega todos los nombres de las opciones
Otros	agregarSinDuplicados/3	Agrega a una lista sin duplicados

Tabla 4 - Funciones del TDA Chatbot		
Tipo de función	Nombre	Descripción
Constructor	chatbot/6	Construye un chatbot
Modificador	chatbotAddFlow/3	Agrega un flujo al chatbot.
Selector	chatbotGetIDFlow/2	Entrega el id del flujo actual
Selector	chatbotGetID/2	Entrega el id del chatbot
Selector	chatbotGetMsg/2	Entrega el msg de bienvenida del chatbot
Selector	chatbotGetName/2	Entrega el nombre del chatbot
Selector	chatbotGetFlows/2	Entrega la lista de flujos del chatbot
Modificador	chatbotUpdateFlowID/3	Actualiza el ID del flujo actual
Selector	chatbotFlowSearch/3	Entrega las opciones de un flujo en base a su ID
Selector	chatbotFindFlow/3	Busca un flujo en base a su ID

Tabla 5 - Funciones del TDA System		
Tipo de función	Nombre	Descripción
Constructor	system/4	Construye un sistema
Modificador	systemAddChatbot/3	Añade un chatbot al sistema
Modificador	systemAddUser/3	Registra un usuario en el sistema
Modificador	systemLogin/3	Loggea a un usuario al sistema
Modificador	systemLogout/2	Desloggea a un usuario del sistema
Modificador	systemTalkRec/3	Permite interactuar con un chatbot
Selector	systemSynthesis/3	Entrega una síntesis del historial de un usuario
Selector	systemGetName/2	Entrega el nombre del sistema
Selector	systemGetChatHistory/2	Entrega el chat history
Selector	systemGetLoggedUser/2	Entrega el usuario logeado
Selector	systemGetChatbots/2	Entrega los chatbots
Selector	systemGetRegisteredUsers/2	Entrega los usuarios registrados
Selector	systemGetChatbotID/2	Entrega el ID del chatbot actual
Selector	systemFindUserChat/3	Accede al historial de un usuario específico
Selector	systemChatbotSearch/3	Entrega el nombre y mensaje de bienvenida del chatbot
Selector	systemFindChatbot/3	Encuentra un chatbot por su ID
Modificador	systemUpdateChatbotID/3	Actualiza el ID del chatbot actual
Modificador	systemUpdateChatbotList/3	Actualiza el ID de un flujo de un chatbot
Modificador	systemAddChatHistory/3	Agrega mensajes de un usuario al sistema
Modificador	systemAddChatbotMsg/3	Agrega mensajes de un chatbot al sistema
Pertenencia	systemUserIsRegistered/2	Comprueba si un usuario está registrado