DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Paradigmas de Programación

Proyecto Semestral de Laboratorio:

Paradigma Lógico

Autor: Sebastián Paillao

Profesor: Edmundo Leiva

Fecha de entrega: 13 de noviembre, 2023

Índice

1-. Introducción

El presente informe corresponde al ramo Paradigmas de Programación, en el cual se plantea un proyecto semestral de laboratorio que es solicitado entregar en diversos lenguajes de programación y siguiendo distintos paradigmas. El paradigma desde el cual se desarrollará la solución en esta entrega corresponde al **paradigma lógico,** y se utilizará el lenguaje de programación Prolog.

Sobre la estructura del informe, éste posee la presente introducción con sus subsecciones para describir el problema, una descripción del paradigma con sus conceptos relacionados y los objetivos a plantear para el desarrollo de la entrega. Posteriormente se tiene el desarrollo junto a sus subsecciones análisis del problema, diseño de la solución, aspectos de implementación, instrucciones y resultados y autoevaluación final sobre el desarrollo de un sistema de chatbots.

Un chatbot es un programa diseñado para simular conversaciones con una persona, generalmente a través de Internet. Puede ser avanzado, utilizando inteligencia artificial para respuestas más complejas y adaptativas a la necesidad de cada usuario, o simple, basado en reglas predefinidas para respuestas específicas, limitaciones que tendrá el sistema a desarrollar. Se utilizan comúnmente en servicio al cliente, asistencia en línea y en diversas aplicaciones de mensajería[1].

1.1-. Descripción del problema

El proyecto aborda el desarrollo de un sistema destinado a la creación, implementación y gestión de chatbots de tipo ITR (Respuesta de Interacción a Texto). Este tipo de chatbots se caracteriza por su estructura definida, respondiendo a consultas de los usuarios mediante opciones preestablecidas, sin procesar lenguaje natural de manera compleja ni emplear inteligencia artificial.

El sistema permitirá a los usuarios registrarse y participar activamente, interactuando con diversos chatbots que forman parte del entorno. Estas interacciones se basarán en respuestas textuales claras y concisas por parte del chatbot. Un aspecto clave del sistema es la capacidad de interconexión entre diferentes chatbots, lo que posibilita un flujo de diálogo más amplio y adaptado a las necesidades de los usuarios. Además, el sistema dará la opción a los usuarios de obtener un historial de sus interacciones, mejorando la experiencia y permitiendo un seguimiento de sus consultas y respuestas recibidas.

Estos chatbot deben tener sus opciones y flujos definidos, dándole al usuario la posibilidad de interactuar con ellos y recorrer los flujos disponibles. Adicionalmente se podrá agregar flujos, opciones o chatbots al sistema, que aporten a la diversidad de las interacciones y aumenten las posibilidades y variedad de respuestas que el usuario podría obtener.

1.2-. Descripción del Paradigma:

El paradigma lógico, en términos generales y entendibles, se basa en la lógica matemática y se distingue por su enfoque en definir "qué es algo" en lugar de "cómo se hace algo". A diferencia de otros paradigmas que se centran en las instrucciones paso a paso, el paradigma lógico utiliza declaraciones lógicas, también conocidas como predicados, para describir hechos y reglas sobre problemas dentro de un sistema de lógica formal. Estas declaraciones

permiten al programa procesar las solicitudes basándose en un conjunto predefinido de reglas y hechos, llevando a una conclusión lógica. Este enfoque es particularmente útil en campos como la inteligencia artificial y el aprendizaje automático, donde la capacidad de razonar sobre conocimientos y deducir nuevas informaciones es esencial[2]. A continuación se definen conceptos esenciales para su entendimiento:

Hechos: Son afirmaciones simples que determinan información fundamental sobre un dominio y son considerados como afirmaciones verdaderas. Por ejemplo, "Mi nombre es Sebastián" podría ser un hecho en un programa lógico.

Predicado: En programación lógica, un predicado es una función que determina la relación entre diferentes términos.

Reglas: Describen las circunstancias bajo las cuales una relación es válida. Contienen un encabezado y un cuerpo y toman la forma "'x' es verdadero si 'y' y 'z' son verdaderos"

Declaratividad: La declaratividad en la programación lógica se refiere a especificar el resultado deseado más que detallar cómo lograrlo. Se centra en el "qué" en lugar del "cómo".

Backtracking: Es un mecanismo para explorar sistemáticamente las posibles soluciones de un problema, retrocediendo cuando se encuentra con un camino sin salida y así volviendo a recorrer el programa o predicado hasta que se cumplan las condiciones necesarias establecidas.

Consulta: Una consulta en programación lógica es una solicitud de información o una pregunta formulada al sistema.

Unificación: Proceso para hacer coincidir variables y constantes en las reglas y hechos, facilitando la deducción de conclusiones, es darle un valor o contenido a una variable.

1.3-. Objetivos

El objetivo del desarrollo de este proyecto es completar el aprendizaje del paradigma funcional, usando el lenguaje Prolog. Para este lenguaje es común usar su propio IDE (SWI-Prolog), pero en este caso se usará el editor de código Visual Studio Code. Este aprendizaje se llevará a cabo desarrollando las soluciones a las diversas problemáticas planteadas en el enunciado del proyecto sobre la creación de un sistema de chatbots, que se espera concluir en su totalidad logrando un funcionamiento perfecto en función de los requerimientos establecidos.

2.- Desarrollo

En esta sección se plasmará el proceso del desarrollo de la solución para el problema, desde el análisis de éste mismo hasta los resultados finales.

2.1.- Análisis del problema

Como requerimientos funcionales, se deben crear los TDA necesarios para poder implementar los predicados requeridos, cada TDA debe estar estructurado en archivos separados de los otros existentes, para luego ser llamado cada uno dentro del archivo 'main' al momento de compilar el programa. Los TDA requeridos, además de sus predicados solicitados son los siguientes:

TDA: Option

Constructor 'option', debe tener como entrada las siguientes variables:

Code (int), *Message* (string), *ChatbotCodeLink* (int), *InitialFlowCodeLink* (int), *PalabrasClave* (list), *Option* (list).

TDA: Flow

Constructor 'flow', debe tener como entrada las siguientes variables:

Id (int), *Message* (string), *Options* (list).

Modificador flowAddOption: Agrega una opción a un flujo, verificando que no pertenezca ya al flujo y retornando false si ya está incluida. Debe tener como entrada las siguientes variables:

Flow (list), *Option*, (list), *NewFlow* (list).

TDA: Chatbot

Constructor 'chatbot', si dentro de los flujos hay duplicidad, agrega sólo uno de los duplicados. Debe tener como entrada las siguientes variables:

ChatbotID (int), *Name* (string), *WelcomeMessage* (string), *StartFlowId* (int), *Flows* (list), *Chatbot* (list).

Modificador 'chatbotAddFlow': Agrega un flujo a un chatbot, verificando que no pertenezca ya al chatbot y retornando false si ya está incluido. Debe tener como entrada las siguientes variables:

Chatbot (list), *Flow* (list), *NewChatbot* (list).

TDA: System

Constructor 'system' debe tener como entrada las siguientes variables:

Name (string), *InitialChatbotCodeLink* (int), *Chatbots* (list), *System* (list).

Modificador 'systemAddChatbot' Agrega un chatbot al sistema, retornando false si ya está incluido. Debe tener como entrada las siguientes variables:

System (list), *Chatbot* (list), *NewSystem* (list).

Modificador 'systemAddUser' Registra un nuevo usuario en el sistema, retornando false si ya hay un usuario con el mismo nombre. Debe tener como entrada las siguientes variables:

System (list), *Username* (string), *NewSystem* (list).

Predicado 'systemLogin' Permite al usuario iniciar sesión, retornando false si ya hay una sesión iniciada por otro usuario o si el usuario no existe. Debe tener como entrada las siguientes variables:

System (list), *Username* (string), *NewSystem* (list).

Predicado 'systemLogout' Cierra la sesión del usuario actual después de verificar que haya un usuario con su sesión iniciada. Debe tener como entrada las siguientes variables:

System (list), *NewSystem* (list).

Predicado 'systemTalkRec', permite al usuario interactuar con el sistema, ingresando una opción numérica por su código o una palabra clave para encontrar la opción que la contiene, en caso de no coincidir con ninguna opción retornar false. Debe tener como entrada las siguientes variables:

System (list), Message (string), NewSystem (list).

Predicado 'systemSynthesis', imprime en pantalla el historial de interacciones entre cierto usuario y el sistema de chatbots, contenido en chatHistory y formateado para mejor experiencia de usuario. Debe tener como entrada las siguientes variables:

System (list), *Username* (string), *String* (string).

Predicado 'systemSimulate', debe permitir simular interacciones entre dos chatbots del sistema definiendo un máximo de interacciones, debe utilizarse el predicado 'myRandom' para generar números aleatorios que definirán los caminos que tomará esta conversación. Debe tener como entrada las siguientes variables:

System (list), *MaxInteraction* (int), Seed (int), *NewSystem* (list).

Además deben plantearse los TDA user y chatHistory junto a sus constructores y predicados necesarios para un correcto funcionamiento del programa, también en archivos separados del 'main' , debiendo tener al final seis archivos de TDA que serán importados en el archivo principal pasa ser ejecutado desde éste.

2.2.- Diseño de la solución

Antes de comenzar a escribir el código del programa requerido, se debe planificar la estructura y el funcionamiento de la solución que se llevará a cabo. Para esto se plantea el diagrama (ver Anexo 1), en el cual se representa cada uno de los TDA y su relación como listas enlazadas. Cada Tipo de Dato Abstracto tiene su estructura y algunos son contenedores de otros, en el diagrama, por ejemplo, puede observarse que chatbot y user pertenecen a la estructura de system, ya que los sistemas a crear contendrán a los chatbots y usuarios, además de los otros parámetros necesarios para establecer las relaciones entre el resto de los TDA. Puede observarse también en el diagrama que los chatbots son contenedores de flujos, y los flujos a su vez contienen opciones. Además, el user, que es contenido por el sistema, también contendrá dentro de su estructura a su historial de interacciones con el sistema, necesario para completar la funcionalidad de los predicados requeridos.

Se plantea además la solución particular de cada uno de los requerimientos funcionales:

TDA: Option

Representación: Option es una lista que contendrá en su estructura el código de la opción, el mensaje, el ChatbotCodeLink, el InitialFlowCodeLink y una lista de keywords o palabras clave relacionadas a esta opción.

Constructor: Se ingresan los datos requeridos, keywords también puede ser una lista vacía, retorna una lista con los elementos ingresados y la estructura propuesta.

El código de las opciones es utilizado para comprobar duplicidad dentro de un flujo al hacer uso del modificador *flowAddOption.*

TDA: Flow

Representación: Flow es una lista que contendrá en su estructura el ID del flujo, su mensaje y una lista de opciones.

Constructor: Se ingresan los datos requeridos, la lista de opciones también puede estar vacía, retorna una lista con los elementos ingresados y la estructura propuesta.

El modificador ***flowAddOption*** utilizará esta estructura de *Flow* para obtener las opciones del flujo y a su vez obtener los códigos de éstas, con el fin de no agregar ninguna opción que comparta código con alguna de las pertenecientes al flujo y retornar *false* en ese caso. En caso contrario, agregará a la o las opciones al flujo.

TDA: Chatbot

Representación: Chatbot es una lista que contendrá en su estructura el ID del chatbot, su nombre, un mensaje de bienvenida, su flujo inicial y una lista de flujos.

Constructor Se ingresan los datos requeridos, la lista de flujos también puede estar vacía, retornará una lista con los elementos ingresados y la estructura propuesta.

El modificador ***chatbotAddFlow*** utilizará esta estructura de *Chatbot* para obtener los flujos del chatbot y a su vez obtener los ID de éstos, con el fin de no agregar ningún flujo que comparta ID con alguno de los pertenecientes al chatbot y retornar *false* en ese caso. En caso contrario, agregará al o a los flujos al chatbot.

TDA: System

Representación: System es una lista que contendrá el nombre del sistema, el InitialChatbotCodeLink, una lista de chatbots y una lista de usuarios.

Constructor: Se ingresan todas las variables requeridas exceptuando la lista de usuarios, que se creará dentro del mismo constructor y se inicializará como lista vacía para contener a futuro a los distintos usuarios a registrar, retornará entonces una lista con toda la estructura mencionada en su representación.

El modificador ***systemAddChatbot*** utilizará esta estructura de *system* para obtener los chatbots del sistema y a su vez obtener los ID de éstos con el fin de no agregar ningún chatbot que comparta ID con alguno de los pertenecientes al sistema y retornar *false* en ese caso. En caso contrario, agregará al o a los chatbots al sistema.

El modificador ***systemAddUser*** utilizará la lista de *Users* del sistema para obtener el nombre de usuario de cada uno de los usuarios, con el fin de evitar crear un nuevo usuario con el mismo username de otro, retornando *false* en ese caso. Si el nombre de usuario ingresado no está en la lista de usuarios, se agregará a ésta, generando también dentro de este user una lista vacía para su futuro historial y un estado de ‘notLogged’, manifestando que su sesión aún no ha sido iniciada.

Se plantea como TDA separado y llamado dentro del predicado *systemAddUser*:

TDA: User

> **Representación:** User será una lista que contiene un nombre de usuario, un historial de chat y un estado de sesión.
>
> **Constructor**: Se ingresa dentro de *systemAddUser* el nombre del usuario a agregar y se llamará a *user* para crear una lista *User* que contiene el *username*, un historial de chat vacío y un estado de sesión 'notLogged'.

El predicado ***systemLogin*** utilizará la lista de *Users* del sistema para consultar si hay algún usuario con su sesión iniciada, si el estado de sesión de uno de los usuarios está establecido como 'logged', el programa retornará *false*. En caso contrario, se cambiará el estado del usuario al cual corresponde el username ingresado de 'notLogged' a 'logged'.

El predicado ***systemLogOut*** utilizará la lista de *Users* del sistema para consultar si hay algún usuario con su sesión iniciada, si el estado de alguno de los usuarios está establecido como 'logged', se cambiará a 'notLogged', en caso de que ningún usuario esté en estado de sesión iniciada, el programa retornará *false*.

El predicado ***systemTalkRec*** utilizará al *system* para obtener los chatbots de los chatbots sus flujos, de los flujos sus opciones y también de la lista de usuarios usará el historial y el estado de sesión. Primero verificará que haya un usuario con su estado de sesión en 'logged', de lo contrario, retornará *false,* después comparará la entrada del usuario con el código de las opciones del flujo del chatbot actual, al encontrar coincidencia entre el código de las opciones o las keywords y la entrada del usuario, dirigirá al usuario al próximo flujo a seguir y guardará en el historial la interacción con su fecha, hora, mensaje y remitente.

Se plantea como TDA separado y llamado dentro de los predicados *systemTalkRec y systemSynthesis* el siguiente:

TDA: ChatHistory

> **Representación:** ChatHistory es una lista que contiene el texto de la interacción, el remitente del mensaje y la fecha y hora de éste.
>
> **Constructor:** Se llama al constructor dentro de la función *systemTalkRec*, el cual obtiene el texto de la interacción, el remitente, la fecha y hora y lo almacena en ChatHistory para luego ser utilizado en la siguiente función.

El predicado ***systemSynthesis***, utilizará el *chatHistory* del user al cual pertenece el username ingresado para imprimir en pantalla las interacciones del usuario en un formato establecido para ser más entendible y agradable a la vista por el usuario.

2.3.- Aspectos de implementación

Para el desarrollo de esta problemática se hizo uso del editor de texto y compilador Visual Studio Code, utilizado por costumbre y comodidad. Se utilizó con una extensión para el uso del lenguaje Prolog y se hizo uso sólo de funciones nativas del lenguaje mencionado, sin

incluir bibliotecas externas y recurriendo a la recursividad para manejar los diversos casos que puedan presentarse en las entradas.

Como pudo plantearse en el diagrama (ver Anexo) y en el diseño de la solución, el programa utiliza listas enlazadas contenidas unas dentro de otras para mantener una estructura sólida y accesible utilizando los predicados y selectores establecidos.

Dentro de la carpeta a la cual pertenece el informe actual se podrá encontrar la carpeta de código fuente, dentro de la cual se plantea cada TDA en un archivo distinto, todos siendo llamados en el archivo *main* para funcionar en conjunto con el script de pruebas perteneciente a la misma carpeta, el cual es detallado en la siguiente sección.

2.4.- Instrucciones de uso

Como ejemplos de uso para el usuario se presenta un script de pruebas, el cual es el archivo 'pruebas_20937236_PaillaoEspindola.pl' dentro de la carpeta de código fuente. En éste archivo de pruebas se podrá encontrar el script de pruebas planteado en el enunciado del laboratorio, y debajo puede encontrarse el script de pruebas provisto por el desarrollador (quien también redacta este informe). En este script de pruebas se plantea el uso de cada una de las funciones por lo menos tres veces cada una, se pueden ver comentados con un '%' los casos en que el programa debe retornar *false*. Para probar el script hasta cierto punto sólo debe copiarse hasta la línea en que quieren testearse los predicados, cambiar la última coma ',', por un punto '.', e ingresar lo copiado como una consulta después de compilar el archivo *main* perteneciente a la misma carpeta. Para probar los casos que deben retornar false debe seguirse la misma lógica, pero descomentando el predicado para que pueda ser tomado en cuenta dentro de la consulta.

2.5.- Resultados y Autoevaluación

Los resultados obtenidos son en su mayoría positivos, exceptuando la implementación de los últimos predicados *systemTalkRec* y *systemSynthesis,* que fueron implementados pero no lograron pasar las pruebas planteadas en el enunciado (se dejaron comentados en el código) y *systemSimulate* que no fue implementado. El resto de los requerimientos funcionales y constructores de TDA fueron implementados y funcionan de manera perfecta según lo estipulado en el 100% de los casos, se hicieron las pruebas planteadas en el script de pruebas del enunciado y también se hicieron las pruebas planteadas en el script creado por el desarrollador, siendo en ambas situaciones exitoso y retornando *false* en los casos específicos que debe hacerlo. De las 14 funciones planteadas se logró desarrollar 11 a la perfección, la autoevaluación resulta con un puntaje de 11.5 de los 14 puntos planteados.

3.- Conclusiones

Se puede concluir que se cumplieron en su mayoría los objetivos establecidos, se logró desarrollar el aprendizaje de uso y manejo del paradigma funcional mediante el lenguaje Prolog. Se aprendió sobre el funcionamiento y estructuras de este paradigma llevando a cabo el desarrollo del código y finalizándolo, cumpliendo con la mayoría de los objetivos planteados. Se tuvo problemas con la implementación de las últimas funciones debido al tiempo disponible para su desarrollo y la complejidad que conlleva la creación de un predicado que maneje tantos datos de distintos TDA, pero se espera implementar para que complete sus funciones en un futuro. Con el paradigma funcional se pueden establecer situaciones y

manejarlas, pero esto sólo funcionaría bajo el contexto establecido por el desarrollador, por esto se llama paradigma lógico, no es muy dinámico pero sabiendo darle uso y establecer el contexto requerido puede resultar muy útil, como en este caso.

4.- Referencias en APA

1.- IBM. ¿Qué es un chatbot?. Recuperado de https://www.ibm.com/es-es/topics/chatbots

2.- Computer Hope. (2019, June 30). Logic programming. Recuperado de https://www.computerhope.com/jargon/l/logic-programming.htm

5.- Anexos

1.- Diagrama de flujo TDAs.