DEPARTAMENTO DE
INGENIERÍA
INFORMÁTICA
UNIVERSIDAD DE SANTIAGO DE CHILE

Paradigmas de programación Laboratorio 2 (Paradigma Lógico – Lenguaje Prolog)

Estudiante: Elías Zúñiga Tobar

Sección: B-2

Profesor: Gonzalo Martínez

Fecha: 13 de noviembre de 2023

Índice

Introducción

Este informe tiene como objetivo el de aprender, entender y explicar sobre el uso del paradigma funcional a través del lenguaje de programación “Prolog” usando el programa de “SWI-Prolog” para el desarrollo de este laboratorio, en donde se tomarán conceptos que no se aplican en otros paradigmas como en el imperativo. Se dará a conocer la problemática de este trabajo junto con su análisis, el paradigma a que se usará, dar a conocer los conceptos que abarca, explicar la solución que se implementará y que después se evaluará y por último se determinará las conclusiones correspondientes.

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, 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 los flujos.
- **Flujo (Flow):** Los flujos serán aquellos elementos que guardarán las opciones, estos contarán también de una ID, un mensaje entrante.
- **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, 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.

Sin embargo, este problema implicará algunas limitaciones debido al uso del paradigma y lenguaje de programación que se explicará a continuación.

Paradigma: En este trabajo se dará el uso del “Paradigma Lógico” donde su mayor exponente es el lenguaje “Prolog”, el cual pertenece a la programación declarativa y su funcionamiento se basa a través de la lógica booleana (datos como true, false, or, and) donde los símbolos se le llaman términos (Term) y estos son contenidos en hechos y/o reglas.

- **Unificación:** Cuando una variable que no tiene ningún valor pasa a tenerlo vamos a decir que dicha variable ha sido ligada, en caso contrario la variable se encuentra sin ligar o no ligada.
- **Hechos:** Un hecho es un mecanismo para representar propiedades o relaciones de los objetos que se están representando. Los hechos declaran los valores que van a ser verdaderos o afirmativos para un predicado en todo el programa.
- **Reglas:** Hechos que dependen de la veracidad de otros hechos.
- **Recursión:** Prolog permite el uso de la recursividad cuando se están definiendo reglas, esto es útil para definir reglas generales y más flexibles.
- **Átomo:** Nombres de objetos, propiedades, o relaciones. Estos deben empezar en minúscula.
- **Consulta:** Es el mecanismo para extraer conocimiento del programa, donde una consulta está constituida por una o más metas que Prolog debe resolver.

Desarrollo

Análisis del problema:

Se solicitó la implementación de este sistema de chatbots a través del software de programación “SWI-Prolog” el cuál usa el lenguaje lógico “Prolog”, 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. **flowAddOption:** Función modificadora para añadir opciones a un flujo.
4. **chatbot:** Función constructora de un chatbot.
5. **chatbotAddFlow:** 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.
7. **systemAddChatbot:** Función modificadora para añadir chatbots a un sistema.
8. **systemAddUser:** Función modificadora para añadir usuarios a un sistema.
9. **systemLogin:** Función que permite iniciar una sesión en el sistema.
10. **systemLogout:** Función que permite cerrar una sesión abierta.

11. **systemTalkRec:** Función que permite interactuar con un chatbot.
12. **systemSynthesis:** Función que ofrece una síntesis del chatbot para un usuario particular partir de chatHistory contenido dentro del sistema.
13. **systemSimulate:** 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 implementación consiste en la creación de listas de elementos a base de hechos y reglas utilizando los predicados implementados en cada archivo TDA, con este paradigma permite un mejor manejo en lo que es la búsqueda de variables en la mayoría de los casos.

Aspectos de implementación:

Para este laboratorio se usó tanto la versión 9.0.4 de “SWI-Prolog” como la versión online de este. En cada archivo TDA se encuentran funciones que trabajan con los datos correspondientes a su campo (Anexo 3. Sin embargo, debido al poco conocimiento respecto al llamado de archivos, se decidió tomar el archivo TDA_system donde se realizarán las consultas aprovechando de que esta llama a todos los demás TDAs.

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 las consultas), si falta uno de estos el programa devolverá error o false, donde se dirá que una función “X” no está definida.

No se puede agregar opciones, flujos o chatbots que ya fueron agregados con anterioridad, sino, el programa retornará false.

Se puede consultar un sistema vacío, pero para agregar un chatbot y un usuario al sistema, se recomienda seguir el siguiente orden (Anexo 4):

1. option
2. flow
3. flowAddOption
4. chatbot
5. chatbotAddFlow
6. system
7. systemAddChatbot o systemAddUser

Caso contrario, el programa puede retornar false o entrar en un ciclo infinito.

Además, para ver las respuestas completas de Prolog, se recomienda ejecutar en la consulta el comando " set_prolog_flag(answer_write_options,[max_depth(0)]). “

Por último, se recomienda realizar las conusltas al archivo tda_system.

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-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.

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, tales como al consultar una variable mediante un selector, este terminaba en un ciclo de recursiones infinitas. Otra dificultad fue el desconocimiento del cómo realizar ciertas tareas como calcular la fecha y hora o el cómo conseguir la id de un usuario, cosa que se tuvo que ingresar a la página de “swi-prolog.org” para resolverlo.

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

Bibliografía

- Patricia Antoima, Cristina Cachero (2015, agosto 19) *Paradigma Lógico*. Recuperado de https://tuparadigma.wordpress.com/2015/08/19/paradigma-logico/

- Uqbar Foundation (2023) *Unificación y pattern matching*. Recuperado de https://wiki.uqbar.org/wiki/articles/unificacion-y-pattern-matching.html
- Cristian González, Julian Salomón (n. d.) *TUTORIAL DE PROLOG*. Recuperado de https://swish.swi-prolog.org/p/Tutorial%20de%20prolog.swinb

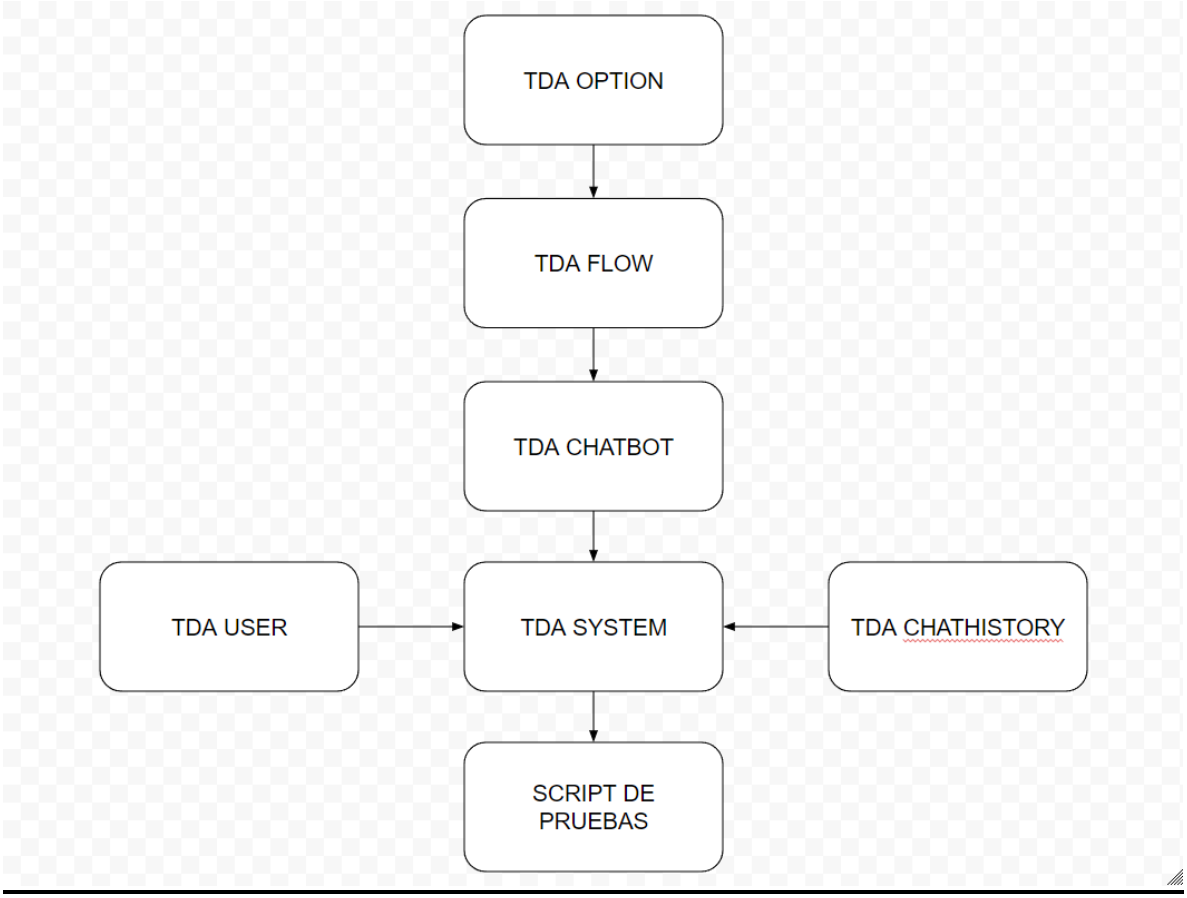
Anexos

1. Tabla de funciones obligatorias

Tipo de Predicado	Nombre de predicado	Funcionamiento
Constructor	option	Crea una lista compuesta de: la ID, un Mensaje, la ID del chatbot perteneciente, el código del flujo inical y una lista con las palabras clave.
Constructor	flow	Primero verifica si se cumple el predicado “detectorOptionsDuplicados”, luego crea una lista compuesta de: ID, un mensaje y una lista con las opciones.
Modificador	flowAddOption	Llama a crear una versión actualizada del flujo de entrada con las nuevas opciones de entrada
Constructor	chatbot	Primero verifica si se cumple el predicado “detectorFlowsDuplicados”, luego crea una lista compuesta de: ID, el nombre del chatbot, el mensaje de bienvenida, la ID del flujo inicial y una lista con los demás flujos.
Modificador	chatbotAddFlow	Llama a crear una versión actualizada del chatbot de entrada con los nuevos flujos de entrada
Constructor	system	Primero se calcula la fecha y hora de creación del sistema, luego verifica si se cumple el predicado “detectorChatbotsDuplicados”, luego crea una lista compuesta de: el nombre del sistema, una lista vacía donde se registrarán los usuarios, un espacio donde se pondrá el usuario que inicia sesión, la ID del chatbot inicial y una lista con los demás chatbots.
Modificador	systemAddChatbot	Primero verifica si se cumple el predicado “detectorChatbotsDuplicados”, luego llama a crear una versión actualizada del sistema de entrada con los nuevos chatbots de entrada.

Modificador	systemAddUser	Primero verifica comparando la ID del usuario de los demás miembros registrados, luego llama a crear una versión actualizada del sistema de entrada con el nuevo usuario de entrada.
Modificador	systemLogin	Verifica si el usuario de entrada está registrado y que no haya alguno usuario conectado para luego crear una versión actualizada del sistema de entrada llenando el espacio de usuario conectado.
Modificador	systemLogout	Verifica si algún usuario inició sesión, para luego crear una versión actualizada del sistema de entrada vaciando el espacio de usuario conectado.
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	Predicado	Descripción
Constructor	putDateTime	Retorna la fecha y hora, en este caso, toma en cuneta la zona UTC, por lo que entrga la hora con 2 horas dem adelanto.
Pertenencia	detectorChatbotsDuplicados	Revisa si al momento de agregar un chatbot al sistema, no haya ningún duplicado. Si no, Prolog devolverá false.
Modificador	registerUser	Registra el usuario al sistema
Pertenencia	isLogedUser	Verifica si un usuario ha iniciado sesión
Selector	getSystemMembers	Devuelve una lista con los nombres de los usuarios registrados
Selector	getMembersIds	Obtiene una lista de las IDs de los usuarios registrados.
Selector	getMembersUser	Obtiene un usuario deseado dentro de los que están registrados.

3.2. TDA-option:

Tipo	Predicado	Descripción
Selector	getOptionId	Obtiene la ID de una opción
Selector	getOptionsIds	Obtiene las IDs de un grupo de opciones

3.3. TDA-flow:

Tipo	Predicado	Descripción
Selector	getFlowsIds	Se obtiene todas las IDs de las opciones dentro de un flujo
Selector	getFlowId	Obtiene la id de un flujo
Pertenencia	detectorOptionsDuplicados	Revisa si al momento de agregar una opción al flujo, no haya ningún duplicado. Si no, Prolog devolverá false.

3.4. TDA-user:

Tipo	Predicado	Descripción
Constructor	user	Se crea una lista con el usuario y una lista vacía para su historial de chat.
Selector	getUserId	Obtiene el ID del usuario, pero para ello, primero trasnforma nombre de usuario en un string de elementos y se lo entrega a “sacarNumeroDeNombre”.
Pertenencia	sacarNumeroDeNombre	Obtiene el número de la lista entregada por “getUserId”.
Selector	getUserName	Obtiene el nombre del usuario.
Selector	getUserHistory	Obtiene el historial del usuario.

3.5. TDA-chatbot:

Tipo	Función	Descripción
Selector	getChatbotsIds	Obtiene la IDs de un grupo de chatbots.
Selector	getChatbotId	Obtiene la ID de un chatbot.
Pertenencia	detectorFlowsDuplicados	Revisa si al momento de agregar un flujo al chatbot, no haya ningún duplicado. Si no, Prolog devolverá false.

3.6. TDA-chatHistory:

Tipo	Predicado	Descripción
Modificador	recMessage	Actualiza el historial del usuario agregando el nuevo mensaje al final de la lista.
Pertenencia	isHistoryNull	Indica si el historial está vacío.
Pertenencia	isHistoryNotNull	Indica si el historial no está vacío.

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

Ejemplo de creación de chatbot

```
option(1, "1) Viajar", 2, 1, ["viajar", "turistear", "conocer"], OP1),
option(2, "2) Estudiar", 2, 1, ["estudiar", "aprender", "perfeccionarme"], OP2),
flow(1, "flujo1", [OP1], F10),
flowAddOption(F10, OP2, F11),
option(3, "3) Comprar", 3, 1, ["comprar", "venta", "negocios"], OP19),
option(4, "4) Cocinar", 4, 1, ["cocinar", "comida"], OP20),
option(5, "5) Leer", 5, 1, ["leer", "lectura"], OP21),
flowAddOption(F11, OP19, F12),
flowAddOption(F12, OP20, F13),
flowAddOption(F13, OP21, F14),
chatbot(0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1, [F14], CB0), %solo añade una ocurrencia de F11
```

Salida ejemplo de chatbot

```
CB0 =
[0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1,
[
[1, "flujo1",
[[4, "4) Cocinar", 4, 1, ["cocinar", "comida"]], [2, "2) Estudiar", 2, 1, ["estudiar", "aprender", "perfeccionarme"]],
[1, "1) Viajar", 2, 1, ["viajar", "turistear", "conocer"]], [3, "3) Comprar", 3, 1, ["comprar", "venta", "negocios"]],
[5, "5) Leer", 5, 1, ["leer", "lectura"]]]
]
]
]
,
```

5. Autoevaluación requerimientos no funcionales:

Requerimientos No Funcionales	Puntaje

Autoevaluación	1
Lenguaje	1
Verisón	1
Standard	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-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-synthesis	0 no se definió la función
system-simulate	0 no se definió la función