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

Informe Laboratorio 2
Paradigma Lógico
Sistema de creación, despliegue y administración de chatbots

Stephan Paul Ramirez

Fecha:

12 de noviembre 2023

UdeSantiago de Chile

Índice de Contenidos

UdeSantiago de Chile

1. INTRODUCCIÓN

En el presente informe se documenta el proceso de realización del segundo laboratorio de la asignatura paradigmas de programación. El objetivo del laboratorio consiste en implementar un ambiente contenedor de chatbots en prolog, usando el paradigma lógico, que permita a un usuario crear e interactuar con distintos chatbots, en donde cada uno contendrá a su vez distintas preguntas de distintos temas y distintas opciones asociadas a otros chatbots, así de esta forma se pueden enlazar las opciones con otras preguntas y así simular una conversación con un usuario. Las operaciones que se abarcan con respecto a crear un chatbot son; identificar un chatbot, añadir preguntas al mismo, añadir opciones a las preguntas y especificar enlaces para el flujo de interacción dentro del chatbot hacia otros chatbots. Luego con respecto a las interacciones del chatbot se tiene que mediante una serie de opciones el usuario pueda escoger alguna de estas para seguir la conversación. También se considera como operación la posibilidad de mostrar una síntesis de la conversación o historial de algún usuario.

2. Descripción del problema

A partir de lo anteriormente mencionado, es que el problema que se plantea en el laboratorio consiste en cómo se podría implementar una idea de solución para crear el sistema contenedor de chatbots usando el paradigma lógico en el lenguaje de prolog, luego esto llevaría plantearse una serie de preguntas: ¿Cómo se podría "traducir" la idea de solución diseñada en el laboratorio 1 a este lenguaje? ¿Cómo se podría implementar el uso de lista de listas para representar los chatbots?, sin embargo, debido a que como se cuenta con los ejemplos de uso y los requerimientos de el enunciado es que se puede contestar a que se deben entonces implementar predicados siguiendo la lógica del primer laboratorio en prolog adaptándose a los nuevos requerimientos usando las herramientas que ofrece prolog, ya sea eliminar duplicados entregando false cuando se encuentra uno en algún constructor o usar la cabeza y cola de las listas para operar estas, etc.

3. Descripción del paradigma

El paradigma lógico es el paradigma con el cual se aborda el proyecto, este consiste en basar la programación en el uso de predicados, es decir, de una manera declarativa, en otras palabras, es centrarse en el que y no en el cómo usando la lógica matemática. Por otra parte, este paradigma se basa en la fórmula "algoritmos = lógica + control" (la llamada Ecuación Informal de Kowalski), lo que significa que un algoritmo se crea especificando conocimiento mediante axiomas (lógica) y el problema se resuelve

mediante un mecanismo de inferencia que actúa sobre el mismo (control) (Merino, 2022).

En el contexto del proyecto este paradigma se emplea usando el lenguaje Swi-Prolog, el cual es una implementación del lenguaje prolog, que está fuertemente basado en el paradigma lógico, un programa en prolog se compone de un conjunto de hechos (afirmaciones simples) y de reglas (que sirven para afirmar la veracidad de un hecho en base a otros). El conjunto de hechos de un programa viene a ser el equivalente a contar con una base de datos, aunque en este contexto se habla de "base de conocimientos", junto con que prolog usa el proceso de unificación en esta base de datos para buscar términos que cumplan una consulta en caso de que algún termino este en mayúscula, que indica que es una variable, usando back tracking automático (Merino, 2020).

Luego como el problema implica almacenar cosas y enlistarlas, se aplicarán los conceptos de listas de prolog para abordar el problema, siendo estos usar la cabeza de una lista y su cola para manejar los predicados.

4. ANÁLISIS DEL PROBLEMA

Identificando los principales componentes de lo que debería ser la implementación de la solución se puede determinar que usando como estructura base las listas, se deberá implementar un sistema "global" que sea una lista de listas que contenga todos los chatbots del sistema, en este contexto se deberán usar predicados que construyan estas estructuras enlistándolas, y teniendo en cuenta que dentro de los requerimientos se especifica que cada estructura deberá tener un id único para su conjunto, eso quiere decir que dentro de un flujo por ejemplo, todas las opciones deberán tener un id único, esto implicaría en prolog entregar false si se ingresan elementos con los mismos identificadores.

Empezando por el primer RF, como se tienen distintas estructuras con distintas operaciones, se deberá implementar una manera de plasmar en el código cada una de estas por separado, en este sentido se deben especificar distintos tipos de datos abstractos en donde cada uno tendrá su representación, constructor, selectores, modificadores y otros predicados asociados. Luego como se mencionó anteriormente, se deberán implementar predicados que sigan estas operaciones de cada TDA siguiendo las reglas de duplicados, es decir, que los constructores devuelvan false si es que existen, lo mismo para los modificadores del estilo "add", junto con que también como se debe interactuar con un usuario, se debe implementar una manera de que un usuario se registre en el sistema, inicie sesión, cierre sesión y que contenga su historial en el mismo.

UdeSantiago de Chile

5. DISEÑO DE LA SOLUCIÓN

Empezando por los TDA de cada estructura, simplemente se siguió la estructura vista en clases para especificar cada uno, esto implica indicar en archivos separados para cada tipo de dato en forma de comentario el nombre de las operaciones, luego en la implementación se agregan estas operaciones como los selectores que son sinónimos de operaciones con la cabeza y la cola de listas. Para el predicado constructor de option se optó por hacer un predicado en donde el ultimo termino es una lista que contiene todos los anteriores, después para verificar unicidad de cada opción en un flujo, se declaró otro predicado que verifica que el id de cada opción es distinto que el de los demás de la lista de options usando una relación recursiva y "maplist" y si no cumple entrega false, luego en el constructor flow se trabaja de manera similar a option pero haciendo uso del predicado anteriormente mencionado. Después para agregar una opción a un flujo, se opta obtener todos los componentes del flujo con sus selectores para luego usarlos en el constructor de flujo, pero con la lista de opciones con la nueva opción. De manera similar se trabaja con chatbot pero como añadir un flujo requería recursividad e insertar al final lo que se hace es usar un predicado que se llama así mismo sin la cabeza de la lista de options pero que verifica en cada llamada recursiva si es que el id del que se inserta es distinto al de la cabeza. Para system, se trabaja de manera similar para verificar unicidad de chatbots, pero ya que la implementación de este puede tener más libertad, se optó por agregar 2 elementos más aparte de los argumentos recibidos, siendo estos la lista con el usuario que inicio sesión y la lista de los historiales de los usuarios, en donde los historiales corresponden al TDA chatHistory que se implementó con una lista de 2 elementos siendo un TDA usuario que es un string realmente y otro que es el registro de su historial que es un string formateado para usar "write" en "systemSynthesis" y por otro lado el otro elemento agregado es la interacción actual que corresponde al último flujo enlazado por el mensaje del usuario, siendo una lista del id del chatbot que contiene al flujo, el id del flujo y el mensaje que lo enlazo. (ver imagen 1)

Continuando con "systemAddUser", se hizo un predicado que agrega un chatHistory con el usuario recibido y un historial vacío en la lista de los historiales. Luego para "systemLogin" y "systemLogout", se optó por reemplazar el apartado de la sesión actual con el nombre del usuario ingresado para iniciar y para cerrar se borra y se deja vacío. Para el predicado "systemTalkRec", la implementación consiste en verificar que el mensaje del usuario haga match con algún option del flujo actual y luego vincular el nuevo flujo reemplazando actual por una lista del "chatbotcodelink" e "initialflowcodelink" de la opción con la cual hizo match el mensaje del usuario, por último, "systemSynthesis", se busca en los historiales el chatHistory del usuario y entrega su registro el cual se actualiza en "systemTalkRec" con cada interacción.

UdeSantiago de Chile

6. ASPECTOS DE LA IMPLEMENTACIÓN

Con respecto a la estructura del proyecto, primero se abordaron los requerimientos funcionales, empezando por los TDA, se documentó en forma de comentario las operaciones de cada uno y luego se empezó a trabajar los predicados de manera ordenada, es decir de 1 en 1 cada día y realizando los commit correspondientes por cada avance en GitHub, junto con que se separaron los TDA en archivos distintos luego de tener al menos los constructores de cada uno, los cuales importan el archivo del TDA que necesitan utilizar, junto con terminar haciendo una archivo "main" que contiene "systemTalkRec" y "systemSynthesis" solamente. Después de terminar la parte funcional, se trabajó en el script de pruebas, de manera que al final se organizó una carpeta llamada "codigofuente" que contiene los TDA, "main" y el script para cumplir con el formato solicitado (cabe destacar que cada archivo también tiene el formato de nombre "nombre_rut_apellidos.extension"). Luego se realizó la autoevaluación, para finalmente hacer el informe del proyecto. Con respecto a el lenguaje y versión utilizados, se usó SWI-Prolog de escritorio, versión 9.0.4

7. INSTRUCCIONES DE USO

Para el uso adecuado del sistema, se deben copiar y pegar consultas similares a las del script de prueba, o también se pueden consultar las mismas en la versión online, esto implica hacer una consulta que contenga cada predicado que se desea utilizar separados por una coma y al final un punto para enviarla. Cabe recalcar que debido a que la única forma de la implementación de reiniciar una conversación es cerrando sesión e iniciar con otra, si es que se inicia sesión con uno antiguo y había una conversación antes, este debe empezar la conversación de 0. (ver imagen 2)

8. RESULTADOS Y AUTOEVALUACIÓN

TDAs	option	flow	flowAddOption	chatbot
1	1	1	1	1
chatbotAddFlow	system	systemAddChatbot	systemaAddUser	systemLogin
1	1	1	1	1
systemLogout	systemTakRec	systemSynthesis	systemSimulate	
1	0.75	1	0	

Con respecto al 0.75 de "systemTalkRec", esto es debido a lo mencionado anteriormente de hablar con un usuario que ya tenía un historial, este debe empezar la conversación de 0 (esto no implica que el historial se borre, se conserva, pero no se puede continuar una conversación de ahí), junto con que para inicializar el sistema

luego de iniciar sesión se debe usar “systemTalkRec” con cualquier cosa como “hola”, debido a que se intentó replicar el resultado del script antes del cambio realizado. El nulo trabajo en “systemSimulate” es debido a que no se logró idear una solución para implementar el predicado en prolog debido a que habría que controlar los números aleatorios que entrega “myRandom” de manera que el sistema no entregue casi todo el tiempo una simulación que termine en false.

9. CONCLUSIÓN

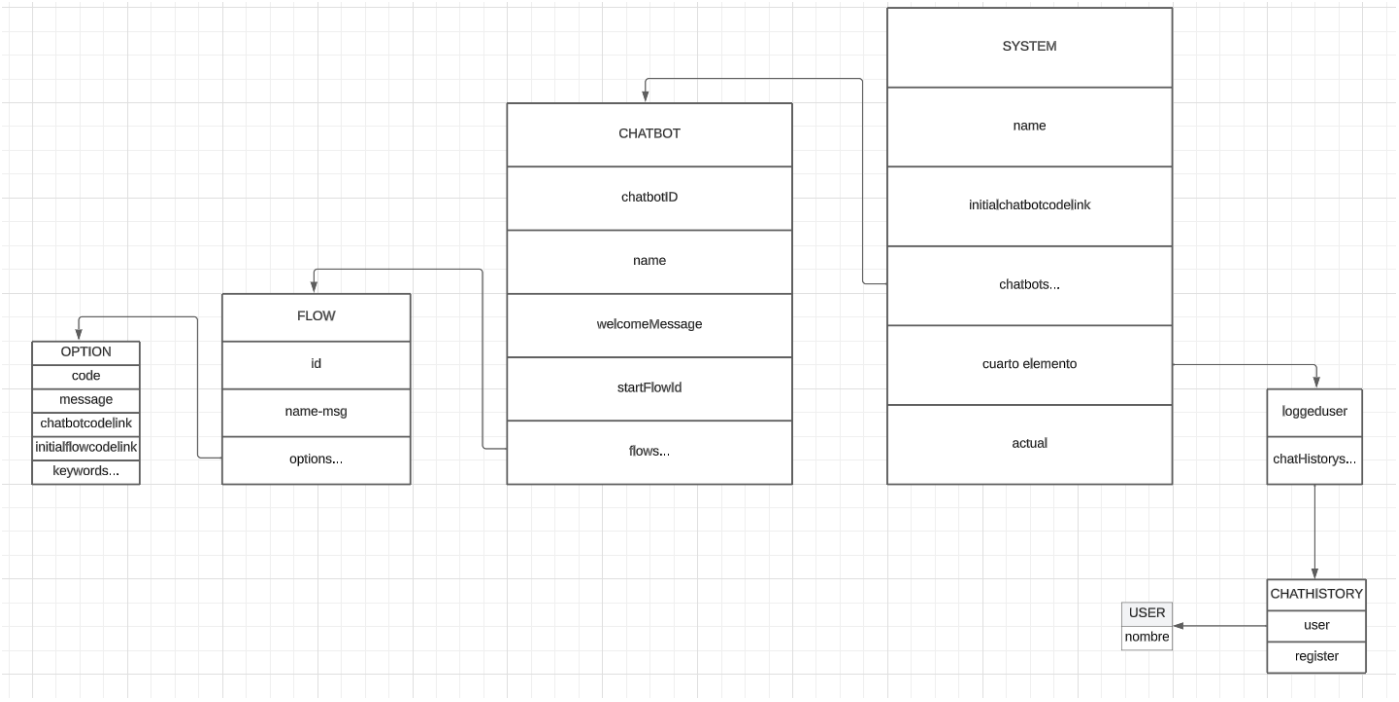
Para finalizar se puede decir que el resultado obtenido cumple con casi todos los requerimientos fundamentales de lo requerido, exceptuando la simulación, ya que el sistema permite crear chatbots con el predicado chatbot, permite añadir preguntas, permite añadir opciones, los enlaces quedan en la misma estructura de una opción y se puede interactuar usando las palabras clave y el numero de la opción. Algunas de las limitaciones del programa son el hecho de que para interactuar con un chatbot requiere hacer una consulta llena de predicados de construcción antes de usar “systemTalkRec”, lo cual dificulta las interacciones con el programa junto con que no se tiene añadido un predicado que vaya mostrando por pantalla la respuesta inmediatamente. Comparando el paradigma lógico con funcional se tiene que debido a que no se pueden utilizar variables, la única forma de conservar datos en funcional es utilizando funciones recursivas o el uso de “let”, lo cual a la larga puede afectar en la legibilidad del código al tener que componer una gran cantidad de funciones, sin embargo, esto se soluciona usando prolog porque a la hora de declarar cláusulas que verifican que algo con mayúscula se cumpla se puede utilizar este como apellido para manipularlo como si se tratara de una asignación, aunque no lo es, simplificando el código, por otro lado, a pesar de que el uso de la unificación de prolog puede ser útil para ahorrar código a veces, un problema de este paradigma es que no se pueden componer funciones como en funcional, así que para hacer operaciones consecutivas se tiene que llamar a una gran cantidad de predicados separados por coma antes de operar lo que a uno le interesa. Un beneficio de este paradigma es que facilita mucho la búsqueda de cosas, así que para “systemTalkRec” se pudo vincular interacciones de manera relativamente sencilla.

10. BIBLIOGRAFÍA

- Merino, M. (2020) *El Lenguaje Prolog: Un ejemplo del Paradigma de Programación Lógica, El lenguaje Prolog: un ejemplo del paradigma de programación lógica*. Available at: https://www.genbeta.com/desarrollo/lenguaje-prolog-ejemplo-paradigma-programacion-logica (Accessed: 12 November 2023).

11. ANEXOS

-Imagen 1



-Imagen 2

```
% ejemplos option:

% se crean con mas de una keyword
option(1, "1) perro", 1, 1, ["perro", "can", "perrito"], Option1),
option(2, "2) gato", 1, 2, ["gato", "felino", "gatito", "michi"], Option2),
% se crean options que mandan al inicio de la conversacion
option(1, "1) calculo", 0, 0, ["calculo"], Option4),
option(2, "2) algebra", 0, 0, ["algebra"], Option5),
option(3, "3) fisica", 0, 0, ["fisica"], Option6),
option(1, "1) grande", 0, 0, ["grande"], Option7),
option(2, "2) chico", 0, 0, ["chico"], Option8),
option(1, "1) corto", 0, 0, ["corto"], Option9),
option(2, "2) largo", 0, 0, ["largo"], Option10),
option(1, "1) Adoptar", 1, 0, ["adoptar"], Option11),
option(2, "2) Estudiar", 2, 0, ["estudiar"], Option12),

% ejemplos flow:

% flow con 2 options
flow(0, "¿Que te gustaria adoptar?", [Option1, Option2], Flow0),
% si se descomenta flow1 da false por option de id repetido
% flow(0, "¿Que quieres estudiar?", [Option4, Option4, Option5], flow1),
flow(0, "¿Que ramo quieres estudiar?", [Option4, Option5, Option6], Flow2),
flow(1, "¿Que tamaño?", [Option7], Flow3),
% se crea flow sin options
flow(0, "¿Que te gustaria hacer?", [], Flow4),
flow(2, "¿Que tan largo el pelo?", [Option9, Option10], Flow5),
```