Universidad de Santiago de Chile
Facultad de Ingeniería
Departamento de Ingeniería Informática

INFORME LABORATORIO Nº2
PARADIGMA LÓGICO
PROLOG

Fecha: 13/11/2023
Nombre: Thomas Gustafsson Cortés.
Profesor: Roberto González.
Sección: A-1.

1. Introducción

El presente informe está compuesto por una descripción del problema a abordar, una descripción del paradigma utilizado, seguido de un análisis del problema en profundidad, un diseño de solución para el problema anteriormente mencionado, aspectos de la implementación, instrucciones de uso, resultados finales y conclusiones. Esto es para lograr abordar el tema del nuevo paradigma utilizado con la respectiva problemática presentada.

Aclarar primeramente que se explicará la implementación de un sistema simplificado que pueda crear, desplegar y administrar un chatbot. Utilizando el paradigma lógico y el lenguaje de programación Prolog.

Siendo el paradigma lógico un tipo de paradigma declarativo, basándose solamente en lógica matemática, utilizando predicados como una base de conocimientos para lograr deducir respuestas a determinadas consultas.

Por otro lado, nace el problema de poder lograr una abstracción únicamente "lógica" de lo solicitado, además de la complicación de poder entender e implementar este nuevo paradigma eficientemente si es que siempre se ha visto y se ha programado anteriormente de forma funcional y no de forma lógica.

2. Descripción del problema

El problema nace desde la necesidad de poder crear un sistema de chatbots ITR (Respuesta de Interacción a Texto) simplificada. El sistema en su fase final debe de poder lograr una fluida comunicación/interacción entre el usuario y el chatbot, y de un chatbot con otro. Siempre girando en todo momento en torno a la utilización del paradigma lógico en Prolog, que es ahí donde recae el verdadero problema, ya que si bien el paradigma facilita en gran manera al momento de utilizar predicados y consultar, pero el nivel de abstracción debe de ser mucho mayor, añadiendo que el punto de vista frente a la necesidad anteriormente planteada, es completamente diferente comparado al paradigma funcional, que fue utilizado para el laboratorio anterior. Si bien este problema es bien limitado al tener en cuenta que solamente se puede utilizar un solo paradigma, el lógico, pero ya que es lo justo y lo necesario que se busca aprender, su alcance cubre por completo el poder lograr entender e implementar bien el paradigma lógico frente a la necesidad de crear un sistema de chatbots ITR.

3. Descripción del paradigma

El paradigma lógico consiste en un tipo de paradigma declarativo donde se basa en gran medida en la abstracción y en todos sus sentidos en la lógica matemática, esto causa que la perspectiva del mundo esté orientado a esto mismo, a lógica. También su principal fuerte es que se centra en el "qué" de los problemas, utilizando una base de datos como el "mundo" que conoce en su totalidad, donde lo que está fuera de ese "mundo" es falso, en la base de datos se declaran los hechos y se crean relaciones entre ellos, mientras que por otro lado, existe también como base fundamental, las consultas a la base de datos, estas consultas son, como dice

el nombre, son consultas, preguntas que se responden en torno al “mundo” creado anteriormente por los hechos ya explicados. Añadir que utiliza dos conceptos en Prolog muy importantes para lograr una búsqueda de elementos, siendo estos la unificación y la inferencia, la primera es que a base de las relaciones de los hechos y atributos se construyan los elementos propiamente necesarios para obtener tal elemento, donde esto ocurre mientras sea verdadero, el segundo sería consultar directamente sobre algo ya conocido. Por último pero no menos importante se encuentra el Backtracking automático que en palabras simples sería un proceso recursivo que revisa en cada uno de los casos hasta lograr una unificación. (Gonzalez, 2023)

4. Análisis del problema

Los requisitos específicos que se deben de cubrir son el de crear un ambiente contenedor de chatbots, crear chatbots donde se puedan identificar, añadir preguntas, opciones y poder enlazarlos mediante flujos hacia otros chatbots. Interacción con el chatbot mediante opciones y textos considerando palabras claves y ofrecer mediante el chatbot una síntesis de las interacciones hechas. Siempre utilizando la mirada del paradigma lógico.

También hay funciones específicos a cubrir, estos son:

- **option:** Crea opciones a escoger, esta construye una opción para un flujo de un chatbot, siendo estos únicos mediante su propio id. Cada opción se une a un chatbot y un flujo específicos.
- **flow:** Función que construye un flujo de un chatbot, siendo este identificado mediante un id, el predicado también verifica que las opciones agregadas no se repitan, esto es verificando por medio de sus id.
- **flowAddOption:** Modifica un flujo para poder añadirle una nueva opción, pero primero verifica que no esté repetida mediante su id, si está repetida la opción, no es agregada y da *false*.
- **chatbot:** Función que crea un chatbot con un id único, se comprueba la duplicidad al añadirlo en un sistema, también este predicado verifica que los flujos añadidos no se repitan comparándolos utilizando el id de cada uno.
- **chatbotAddFlow:** Añade un nuevo flujo a un chatbot, verificando que este no se repita, esto se valida mediante el id del flujo, si es así, no se añade y da *false*. Añadir que se debe implementar un tipo de recursión, en este caso se usó la recursión natural.
- **system:** Función que construye un sistema de chatbots y deja registro de la fecha y hora de la creación. Este también contiene el historial del chat de cada usuario, aclarar que tiene el String formateado de cada mensaje del usuario y chatbot, fecha, hora y emisor. También el predicado de este mismo verifica que los chatbots sean distintos entre sí en base al Id.
- **systemAddChatbot:** Función que añade un nuevo chatbot a la lista de chatbots de un sistema en específico, pero primero verifica que este nuevo chatbot, por medio de su id, no se repita. Si está repetido, da *false*.

- **systemAddUser:** Añade un nuevo usuario al sistema, verificando que no se repita su nombre en otros usuarios, ya que si es así, dará *false*.
- **systemLogin:** Hace iniciar al usuario de sesión en el sistema, pero primero se comprueba que no esté "conectado", que no haya nadie con su misma id (su nombre) o que no existe una sesión ya iniciada por otro usuario.
- **systemLogout:** Permite cerrar la sesión anteriormente abierta por un mismo usuario.
- **systemTalkRec:** Función que permite al usuario interactuar con el chatbot, revisando previamente que el usuario haya iniciado sesión para poder interactuar.
- **systemSynthesis:** Entrega una síntesis del chatbot para un usuario en particular, esto es a partir del historial del chat del usuario que está en el respectivo sistema.
- **systemSimulate:** Permite que exista una simulación de interacción entre dos chatbots. De forma pseudoaleatoria, se da la posibilidad de que interactúen de forma “realista”, si no hay más interacciones, la simulación termina, esto es al igual con la cota superior, que es el número máximo de interacciones.

5. Diseño de la solución

Para empezar a diseñar la solución a la problemática se empezó planteando los TDA’s fundamentales para lograr el objetivo dado, el primero fue el option, la base de todo, ya que este representa las opciones/respuestas que puede optar el usuario ante el chatbot, se constituye por un id único, un mensaje predeterminado que entregar, el link del id del chatbot asociado y del flow inicial y también posee una lista de las palabras claves, que serán necesarias para después. Esto fue necesario crearlo primero para poder crear el TDA flow ya que este se representa como un flujo en particular sin ningún tipo de repetición de un chatbot, este se representa por un id único de tipo entero positivo, un nombre del mensaje propiamente tal del flujo de tipo string y un listado de las posibles opciones a escoger, donde cada uno de los elementos es de tipo option. Siendo necesario verificar cada elemento ingresado para saber si es de acuerdo a lo solicitado y a los parámetros dados, de esta forma se puede lograr añadir opciones nuevas a un flujo, siempre y cuando las Id de las opciones no se repitan, que se puede verificar rápidamente utilizando un predicado *member* propio, ya que es necesario revisar listas en listas, si se repite un Id, dará *false* para así informar de una falsedad en el caso propuesto, este mismo procedimiento y pensamiento lógico se puede emplear en cada predicado similar donde se quiera añadir y verificar por medio de una Id, así que se creará uno general para poder utilizarlo más rápidamente en momentos generales. A continuación se crea el TDA del chatbot como tal, este posee un id único, un nombre, un mensaje de presentación, el id del flow inicial asociado y una lista de los flujos, recordando de que cada elemento en cada TDA está representado por un *integer* positivo, un *string* o de tipo lista de otros TDA’s. De esta misma forma se crea el TDA system donde se alberga todo lo anterior, que por consiguiente es

necesario crear el TDA usuario que representa a un usuario y el TDA de un historial del chat que se representa por una hora y fecha exacta y un mensaje.

Al tener todo ya establecido y representado se puede continuar a la creación del sistema de *login* y *logout* del usuario, donde a grandes rasgos es agregarlo (verificando duplicidad) a la lista de usuarios conectados en el sistema o quitarlo, de esta forma se puede crear el predicado *systemTalkRec*, donde se utilice las palabras claves de cada opción o utilizar sus Id respectivos para escoger una opción, siempre y cuando el usuario esté conectado (verificando por medio de su nombre como Id). Con esto terminado se puede obtener una síntesis de tipo string reseteable para cada conversación de cada usuario. De esta forma ya se podría tener una conversación completa con un sistema completo insertado por medio de la consola de Prolog.

6. Aspectos de la implementación

Para la implementación en este laboratorio se estructuró principalmente de 6 TDA's, siendo estos option, flow, chatbot, system, user y chatHistory. Estos TDA's fueron necesarios para la implementación total del ambiente esperado, ya que estos son los requerimientos principales obligatorios.

Cada uno de estos TDA's poseen sus respectivas representaciones, constructores, selectores, modificadores, otras operaciones y funciones de pertenencia necesarios para un buen y completo uso de estos mismos.

Se utilizó SWI-Prolog de 64 bits, versión 9.0.4 en lenguaje de Prolog, las bibliotecas empleadas fueron solamente las que vienen con Prolog.

7. Instrucciones de uso

Para poder utilizar y ejecutar los comandos y crear un sistema de chatbots, se requiere que se utilice por medio de los comandos, los predicados para conformar las opciones, los flujos, los chatbots y los sistemas como guste, un ejemplo sería el *Script de Pruebas [Figura 1],* de esta forma ya se tiene creado por completo el sistema de chatbots con sus respectivas opciones, palabras, etc. Cabe destacar que hay que escribir todo de corrido con comas como separación entre predicados y de esta forma puede con el respectivo predicado, interactuar con el chatbot correspondiente, y si es necesario, le solicita de la misma forma una síntesis del chat de un usuario respectivo.

8. Resultados

Probando con todos y cada uno de los predicados realizados (hasta *systemLogout*), cumplen su función en un 100%, pero lastimosamente no se pudo lograr finalizar por completo todo lo esperado. Pero cada uno de los predicados hechos fueron probados con cada uno de las posibles complicaciones que podrían tener, como el *chatbotAddFlow* por ejemplo, o el *systemLogin*, que pueden mostrar errores fácilmente.

De los predicados no implementados fueron el systemTalkRec, systemSynthesis y el systemSimulate, que por falta de tiempo y de entendimiento sobre el paradigma para abordarlos, no se logró implementar.

9. Conclusiones

En conclusión me apena decir que no se logró lo esperado, la implementación en su totalidad del sistema de chatbots, pero cabe destacar el uso del paradigma presente, el paradigma lógico, ya que este tiene un gran alcance en poder encontrar las soluciones y resolver consultas ya que el mismo paradigma lo resuelve, pero se necesita un nivel más de abstracción en comparación al paradigma funcional, que si bien es de la misma familia, pero con una mirada completamente diferente entre estos dos. Las limitaciones del paradigma lógico son principalmente lo inflexible que son las reglas y las condiciones, ya que ante la necesidad de hacer un cambio, se debe de tener sumo cuidado y tener en cuenta todo el código para poder cambiar gran parte de los predicados. Pero gracias al backtracking uno se ahorra muchos predicados que hubieran sido necesarios en el paradigma funcional, si bien este paradigma lógico facilita demasiado en algunos casos, pero también tiene sus propias grandes complicaciones.

En conclusión se logró implementar eficientemente este paradigma en cada punto realizado, pero no fue terminado el proyecto en su totalidad, por la falta de familiaridad con el paradigma. Lo más importante ante este paradigma es familiarizarse con la familia declarativa primeramente, y luego abordar el pensamiento lógico, de otra forma sería mucho más difícil el proceso de la elaboración del proyecto.

10. Referencias

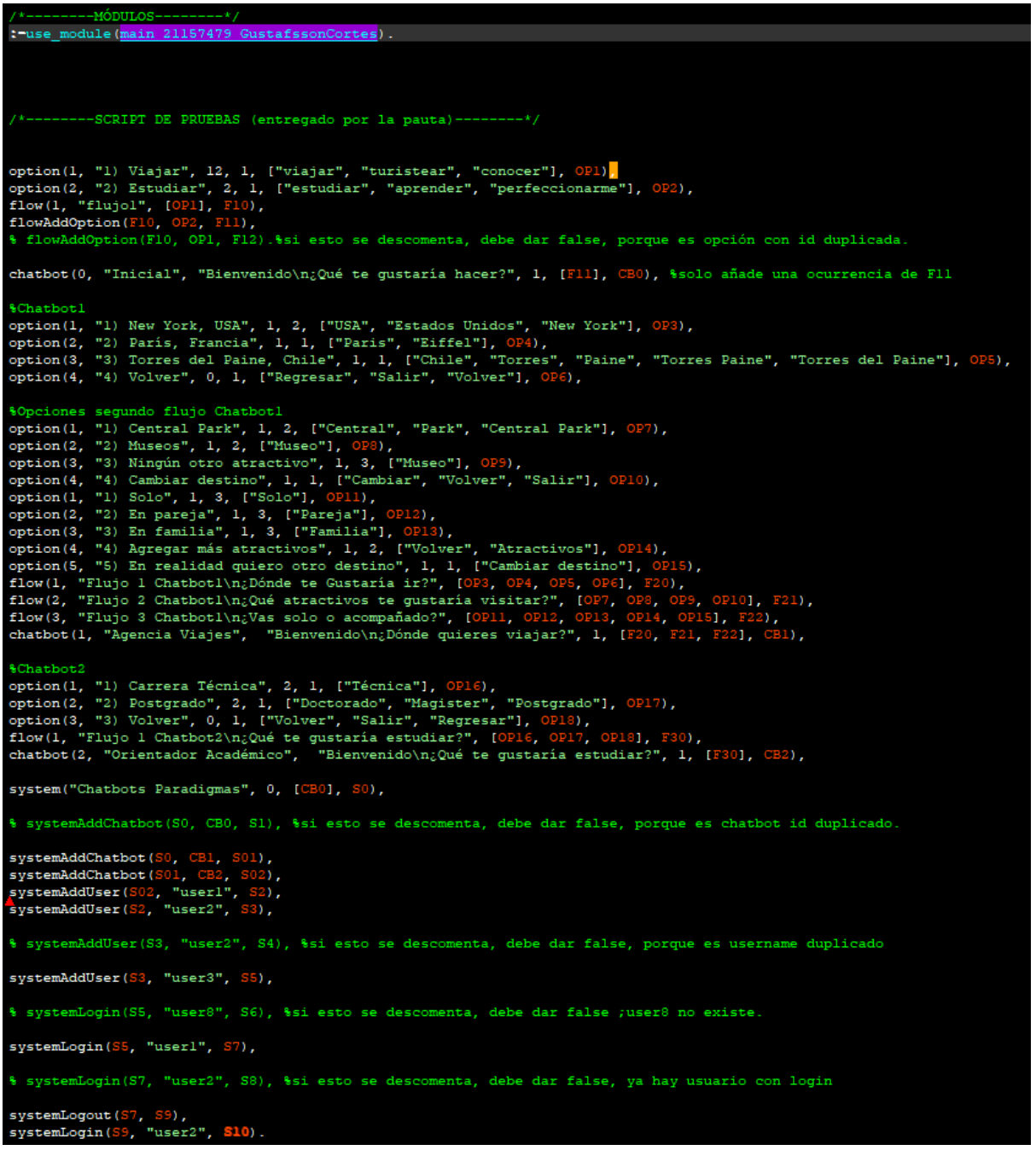
Gonzalez, Roberto. (2023). - Programación Lógica. Diapositivas/videos extraídos desde Moodle Usach - Paradigmas de Programación (13204 y 13310) 2-2023.

“Paradigma logico.” *uqbar-wiki*, https://wiki.uqbar.org/wiki/articles/paradigma-logico.html. Accessed 13 November 2023.

SWI-Prolog.org, https://www.swi-prolog.org. Accessed 13 November 2023.

11. Anexos

[Figura 1]



```
/*--------MÓDULOS--------*/
:-use_module(main_21157479_GustafssonCortes).



/*--------SCRIPT DE PRUEBAS (entregado por la pauta)--------*/


option(1, "1) Viajar", 12, 1, ["viajar", "turistear", "conocer"], OP1),
option(2, "2) Estudiar", 2, 1, ["estudiar", "aprender", "perfeccionarme"], OP2),
flow(1, "flujo1", [OP1], F10),
flowAddOption(F10, OP2, F11),
% flowAddOption(F10, OP1, F12).%si esto se descomenta, debe dar false, porque es opción con id duplicada.

chatbot(0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1, [F11], CB0), %solo añade una ocurrencia de F11

%Chatbot1
option(1, "1) New York, USA", 1, 2, ["USA", "Estados Unidos", "New York"], OP3),
option(2, "2) París, Francia", 1, 1, ["Paris", "Eiffel"], OP4),
option(3, "3) Torres del Paine, Chile", 1, 1, ["Chile", "Torres", "Paine", "Torres Paine", "Torres del Paine"], OP5),
option(4, "4) Volver", 0, 1, ["Regresar", "Salir", "Volver"], OP6),

%Opciones segundo flujo Chatbot1
option(1, "1) Central Park", 1, 2, ["Central", "Park", "Central Park"], OP7),
option(2, "2) Museos", 1, 2, ["Museo"], OP8),
option(3, "3) Ningún otro atractivo", 1, 3, ["Museo"], OP9),
option(4, "4) Cambiar destino", 1, 1, ["Cambiar", "Volver", "Salir"], OP10),
option(1, "1) Solo", 1, 3, ["Solo"], OP11),
option(2, "2) En pareja", 1, 3, ["Pareja"], OP12),
option(3, "3) En familia", 1, 3, ["Familia"], OP13),
option(4, "4) Agregar más atractivos", 1, 2, ["Volver", "Atractivos"], OP14),
option(5, "5) En realidad quiero otro destino", 1, 1, ["Cambiar destino"], OP15),
flow(1, "Flujo 1 Chatbot1\n¿Dónde te Gustaría ir?", [OP3, OP4, OP5, OP6], F20),
flow(2, "Flujo 2 Chatbot1\n¿Qué atractivos te gustaría visitar?", [OP7, OP8, OP9, OP10], F21),
flow(3, "Flujo 3 Chatbot1\n¿Vas solo o acompañado?", [OP11, OP12, OP13, OP14, OP15], F22),
chatbot(1, "Agencia Viajes",  "Bienvenido\n¿Dónde quieres viajar?", 1, [F20, F21, F22], CB1),

%Chatbot2
option(1, "1) Carrera Técnica", 2, 1, ["Técnica"], OP16),
option(2, "2) Postgrado", 2, 1, ["Doctorado", "Magister", "Postgrado"], OP17),
option(3, "3) Volver", 0, 1, ["Volver", "Salir", "Regresar"], OP18),
flow(1, "Flujo 1 Chatbot2\n¿Qué te gustaría estudiar?", [OP16, OP17, OP18], F30),
chatbot(2, "Orientador Académico",  "Bienvenido\n¿Qué te gustaría estudiar?", 1, [F30], CB2),

system("Chatbots Paradigmas", 0, [CB0], S0),

% systemAddChatbot(S0, CB0, S1), %si esto se descomenta, debe dar false, porque es chatbot id duplicado.

systemAddChatbot(S0, CB1, S01),
systemAddChatbot(S01, CB2, S02),
systemAddUser(S02, "user1", S2),
systemAddUser(S2, "user2", S3),

% systemAddUser(S3, "user2", S4), %si esto se descomenta, debe dar false, porque es username duplicado

systemAddUser(S3, "user3", S5),

% systemLogin(S5, "user8", S6), %si esto se descomenta, debe dar false ;user8 no existe.

systemLogin(S5, "user1", S7),

% systemLogin(S7, "user2", S8), %si esto se descomenta, debe dar false, ya hay usuario con login

systemLogout(S7, S9),
systemLogin(S9, "user2", S10).
```