Laboratorio 2 – Paradigma lógico “Prolog”

Estudiante: Tamara Cerna Vargas
Profesor: Víctor Flores

Contenido

1. Introducción

Los lenguajes de programación declarativos se centran en expresar programas de manera más cercana al problema que a los detalles de implementación. Según Lloyd, la declaratividad implica que los programas se consideran teorías en una lógica adecuada y la computación se realiza mediante deducción basada en esa teoría. Un ejemplo destacado de un lenguaje de programación declarativo es Prolog, que se basa en la deducción lógica. Prolog ha demostrado eficiencia en implementaciones que compiten con otros lenguajes de alto nivel. Además, su flexibilidad ha servido como punto de partida para desarrollar nuevas ideas, como la programación con restricciones y enfoques multi-paradigma, que combinan elementos de programación funcional, orientada a objetos e imperativa. (la factoria apple, s.f.)

el informe tiene por objetivo describir el proceso de creación de chatbots, centrados en respuestas predefinidas a preguntas específicas, respetando los fundamentos del paradigma lógico y desarrollando la solución en el lenguaje Prolog.

La estructura del informe consta de introducción, descripción breve del problema, análisis del problema, diseño de la solución, consideraciones de implementación, instrucciones de uso con ejemplos claros de uso, resultados obtenidos, evaluación completa y conclusiones.

2. Descripción de la problemática

La problemática abordada bajo el paradigma lógico se enfoca en la creación de chatbots de Respuesta de Interacción a Texto (ITR), los cuales se caracterizan por su estructura definida y la capacidad de responder a preguntas específicas mediante opciones predefinidas. Estas opciones pueden ser seleccionadas por el usuario a través de palabras clave, sinónimos, números o letras, estableciendo así una interacción estructurada con el chatbot. Sin embargo, debido a su simplicidad, la meticulosidad en la gestión de opciones, flujo y palabras clave es crucial para evitar generar una experiencia fragmentada para el usuario. Entre las desventajas, destaca la falta de adaptabilidad y capacidad de aprendizaje de estos chatbots, lo que limita su eficacia en escenarios más complejos donde las preguntas pueden variar considerablemente. (Payssé, 2019)

3. Descripción del paradigma y conceptos aplicados

El paradigma lógico se caracteriza como un enfoque declarativo basado en la lógica matemática. Los programas se construyen mediante declaraciones y reglas lógicas, haciendo uso de predicados para describir y relacionar individuos en el programa. Prolog emerge como el lenguaje más comúnmente asociado con este paradigma. Se destaca por su naturaleza declarativa, enfocándose en la descripción del problema en lugar de la especificación detallada de una solución. En resumen, el paradigma lógico ofrece un enfoque poderoso y conciso para la resolución de problemas, fundamentado en los principios de la lógica matemática. (Spigariol, 2005)

En el contexto de la problemática tratada, se emplea el paradigma lógico para establecer reglas, dando origen a Tipos de Datos Abstractos (TDA) como Option, Flow, System, Chatbot, Chathistory y User. Dentro de la implementación (Constructores, Funciones de Pertenencia, Selectores, Modificadores y Otras), se logran por medio de metas secundarias, estas extraen la información

necesaria para cumplir la función (meta primaria). Para conectar los distintos elementos del chatbot, se implementan cláusulas y reglas lógicas adicionales, facilitando la creación de funciones esenciales para el funcionamiento del sistema. Este enfoque proporciona una estructura coherente y eficiente para gestionar la interacción del chatbot, asegurando la consistencia y evitando redundancias en la información.

4. Análisis del problema

Dentro de los requerimientos funcionales, se incluye la especificación e implementación de los Tipos de Datos Abstractos (TDA), con el fin de determinar un modelado del problema de forma adecuada. Se definieron seis TDAs correspondientes a Option, Flow, System, Chatbot, Chathistory y User (mencionados en el apartado 3). La estructura de estos es "nombre(Argumentos)". Para Option, el constructor se definió como option(CodeOp, Mensaje, ChatbotCodeLink, InitialFlowCodeLink, Keyword, [CodeOp, Mensaje, ChatbotCodeLink, InitialFlowCodeLink, Keyword]). El dominio corresponde a enteros para CodeOp, ChatbotCodeLink y InitialFlowCodeLink, mientras que string para Mensaje y Keyword (palabras clave o sinónimos). Consta de la meta principal option/6.

Dentro de la capa selectora se tienen las funciones getMensajeOption, getKeywordOption, getChatbotCodeLinkOption y getInitialFlowCodeLinkOption; sus respectivas metas primarias constan de 2 entradas y el objetivo es poder rescatar partes concretas de la estructura Option. Solo cuenta con la capa selectora.

El constructor de Flow es flow(Id, NameMsg, Option, [Id, NameMsg, SinDuplicados]), su dominio es entero para Id, string para NameMsg y una lista de Option, esta lista no debe tener opciones repetidas. Dentro de la capa selectora (getter) se encuentran getIdFlow, getNameMsgFlow y getOptionFlow. Para la realización de la capa modificadora de Flow, se recolectaron mediante los getters los predicados de Flow, y el elemento a modificar se agrega en una nueva estructura de Flow. Solo se cuenta con setOptionFlow(Flow, NewOption, FlowNew). Dentro de los requerimientos se encuentra flowAddOption, función que recibe una opción y verifica si es posible añadirla a la estructura de Flow; para su implementación fue necesario utilizar todo el TDA Flow.

El proceso de construcción de Chatbot es similar a Flow; el constructor se compone de chatbot(ChatbotID, Name, WelcomeMessage, StartFlowId, Flows, [ChatbotID, Name, WelcomeMessage, StartFlowId, SinDuplicados]). Su dominio es entero para ChatbotID y StartFlowId, string para Name y WelcomeMessage, y una lista de Flows. Dentro de los selectores se encuentran getIdChatbot, getNameChatbot, getMessageChatbot, getStartFlowChatbot y getFlowChatbot. Solo cuenta con un modificador setFlowChatbot, el cual utiliza todos los getters mencionados como metas secundarias. Dentro de los requerimientos se encuentra la función chatbotAddFlow, cuya implementación es similar a flowAddOption.

Para la construcción de System se utilizó el constructor system(Users, Name, InitialChatbotCodeLink, Chatbot, [Users, Name, InitialChatbotCodeLink, SinDuplicados]). Sus funciones modificadoras son getUserSystem, getNameSystem, getChatbotCodeLinkSystem y getChatbotSystem. Dentro de las funciones modificadoras se encuentran setChatbotSystem(System, NewChatbot, NewSystem) y setSystemUser(System, NewUser, NewSystem). Dentro de los requerimientos se encuentran las funciones systemAddChatbot y systemAddUser; la lógica es similar a chatbotAddFlow y

flowAddOption. Para la implementación de systemLogin, fue necesaria la creación de systemaLogueado; esta función cuenta con un contador, el cual si es mayor a 2 indica que el usuario está logueado. En este caso, retornará falso. Y para systemLogout, se elimina la primera ocurrencia de la lista de usuarios de System correspondiente al usuario logueado.

Para el logueado del usuario fue necesario hacer el constructor de User y Chathistory(necesario para funciones system-talk-rec, system-synthesis, system-simulate, lamentablemente no se pudieron realizar).

5. Diseño de la solución

Para la creación del constructor, independientemente del TDA descrito, se han utilizado listas, y en algunos casos, listas anidadas para su representación. A modo de ejemplo, se presenta el constructor de Option:

“option(CodeOp, Mensaje, ChatbotCodeLink, InitialFlowCodeLink, Keyword,[CodeOp, Mensaje, ChatbotCodeLink, InitialFlowCodeLink, Keyword])”. Cuyo dominio corresponde a números enteros para CodeOp, ChatbotCodeLink y InitialFlowCodeLink, mientras que cadena de caracteres para Mensaje y Keyword (palabras clave o sinónimos).

En el caso de los selectores (getters) y modificadores (setters), se sigue un enfoque similar. Para los primeros, se accede a la estructura del constructor y se selecciona la parte específica a rescatar. Por ejemplo, “getMensajeOption([_, Mensaje, _, _, _], Mensaje)” corresponde a un selector para obtener el mensaje de un objeto Option.

En el caso de los modificadores, se requieren los selectores de toda la estructura excepto la parte que se va a modificar (metas secundarias). Esta parte se guarda en un nuevo constructor. Un ejemplo sería “setOptionFlow(Flow, NewOption, FlowNew)”, que utiliza “getIdFlow” y “getNameMsgFlow” para construir el nuevo objeto Flow con la modificación realizada: “flow(ID, Msg, NewOption, FlowNew)”.

Dentro de los requerimientos funcionales del Tipo de Dato Abstracto (TDA) `Flow`, se encuentra la función "flowAddOption". Esta función recibe una opción y verifica si es posible añadirla a la estructura de `Flow`. Su implementación requirió el uso de metas secundarias, tales como "getOptionFlow/2", "mimbar/2", "append/3", y "setOptionFlow/3". Para el TDA de `Chatbot`, se encuentra la función "chatbotAddFlow". La lógica de implementación es similar a "flowAddOption", utilizando metas secundarias como "getFlowChatbot/2", "member/2", "append/2", y "setFlowChatbot/3". Por último, dentro del TDA `System`, se encuentran las funciones "systemAddChatbot" y "systemAddUser"; La lógica de implementación para estas funciones es análoga a "chatbotAddFlow" y "flowAddOption", haciendo uso de metas secundarias específicas, como "getFlowChatbot/2", "member/2", "append/2", y "setFlowChatbot/3".

Además de "systemAddChatbot" y "systemAddUser", se incluyen las funciones "systemLogin" y "systemLogout". En la implementación de "systemLogin", fue necesario introducir "systemaLogueado", que cuenta el número de usuarios logueados. Un usuario puede estar presente en un máximo de dos instancias, correspondientes a "systemAddUser" y "systemLogin". Si este número excede dos, la función retorna falso. En relación con "systemLogout", la operación implica

la eliminación de la primera posición de la lista de usuarios en el constructor `System`. Dicha posición representa al usuario logueado, cerrando de esta manera la sesión del sistema.

Para la función "system-talk-rec", que permite la interacción con el chatbot, la idea de implementación es la siguiente. Primero, mediante la función "sistemaLogueado", se verifica la existencia de un usuario en el sistema. Luego, se obtienen los chatbots pertenecientes al sistema, los flows asociados a dichos chatbots, y las opciones correspondientes a esos flows. Se compara el mensaje obtenido por el usuario con las opciones disponibles. Cuando ambas variables coinciden, se rescata el "ChatbotCodeLink" y "InitialFlowCodeLink" de la opción. Se guarda la interacción en el historial del usuario, incluyendo el usuario, el mensaje, el chatbot identificado con "ChatbotCodeLink", y el string del chatbot. A través de la recuperación del "InitialFlowCodeLink", se navega al flujo correspondiente a la opción seleccionada. Para "system-synthesis", un predicado que ofrece una síntesis del chatbot para un usuario particular a partir del chatHistory contenido dentro del sistema, se rescatan los elementos guardados en la estructura de chatHistory y se imprimen mediante la función "write". Por último, "system-simulate" permite generar la simulación en términos de interacciones y utilizando la función “myRandom”.

6. Aspectos de implementación

La estructura del proyecto consta de 6 archivos .pl, correspondientes a los TDA’s (tdaOption_21077657_CernaVargas.pl, tdaFlow_21077657_CernaVargas.pl, tdaChatbot_21077657_CernaVargas.pl, tdaSystem_21077657_CernaVargas.pl tdaUser_21077657_CernaVargas.pl y tdaChatHistory_21077657_CernaVargas.pl).

Los cuales a su vez se estructuran mediante constructores, funciones de pertenencia, selectores, modificadores y otras funciones; todas las funciones esta comentadas con su dominio, meta principal, meta secundaria y descripción. Los scripts por corroborar se encuentran en un archivo .pl, el cual no se ejecuta, y el archivo léeme .txt, que cuenta con las instrucciones para ejecutar los scripts de prueba. Dado a la manera de implementación para verificar el script de prueba (inexistencia de un archivoMain.pl), es que no se utilizó ninguna librería externa.

El entorno de desarrollo integrado (IDE) utilizado es SWI-Prolog versión 9.1.17, además de su versión online, y el lenguaje correspondiente al laboratorio es Prolog.

7. Instrucciones de uso

Se deben descargar los archivos del repositorio, abrir la consola de Prolog y realizar consult para cada archivo con la dirección del archivo, en total deben existir 6 consult, uno por cada TDA. A modo ejemplo: consult('C:/Users/Tamara/OneDrive/Escritorio/Laboratorios Paradigmas/Lab Prolog/codigo fuente/ tdaOption_21077657_CernaVargas.pl’).

dentro del archivo prueba .pl existen 2 script de prueba, el primero entregado en catedra y el segundo diseñado a modo ejemplo. Ya realizado los consult, al lado del signo ? se debe pegar el primer script de prueba, confirmada su ejecución, se pegara el siguiente script de prueba a continuación del mismo signo.

En la sección de anexo se puede ver un ejemplo secuencial del proceso y 2 diagramas correspondientes a los scripts de prueba.

8. Resultados y autoevaluación

Dentro de los resultados obtenidos, se pudo implementar con éxito los requisitos no funcionales, mientras que en los requerimientos funcionales fue posible implementar hasta la función 11. El código paso por constantes cambios llevando a cabo un archivo de texto en el que se dejaba constancia del estado de las funciones implementadas e ideas para los requerimientos futuros, link el archivo se encuentra en anexo. No fue posible seguir avanzando en el laboratorio por carga académica y problemas de lógica. Para información sobre la autoevaluación y su nivel de logro revisar archivo autoevaluacion_21077657_CernaVargas.txt, adjunto en repositorio de Github.

9. Conclusiones del trabajo

En este proyecto, se abordó la creación de chatbots bajo el paradigma lógico utilizando el lenguaje de programación Prolog. El diseño de la solución se basó en la construcción de Tipos de Datos Abstractos (TDA), como Option, Flow, System, Chatbot, Chathistory y User, utilizando listas y reglas lógicas. Se implementaron funciones para añadir opciones a flujos, incorporar flujos a chatbots y gestionar la interacción entre el sistema y los usuarios. En cuanto a los resultados y autoevaluación, se lograron implementar con éxito los requisitos no funcionales y una parte significativa de los requerimientos funcionales, a comparación del laboratorio pasado que solo fue posible lograr hasta la implementación número 9.

Dentro de las limitaciones se reconocen la carga horaria y la falta del material actualizado (script de prueba). En resumen, este proyecto proporcionó experiencia práctica en la aplicación del paradigma lógico y del lenguaje Prolog para abordar problemas específicos, asociados con la creación de chatbots bajo este enfoque declarativo.

10. Referencias

la factoria apple. (s.f.). *la factoria apple*. Obtenido de https://www.lafactoriaapple.com/ciencias-de-la-computacion/lenguaje-declarativo.php

Payssé, M. (29 de octubre de 2019). *InConcert*. Obtenido de https://blog.inconcertcc.com/es/dumb-chatbots-como-los-bots-de-itr-pueden-ayudarte-a-automatizar-la-atencion/

Spigariol, L. (2005). *Fundamentos teoricos de los paradigmas de programación.* Buenos Aires.

11. Anexo

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.1.17)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult('C:/Users/Tamara/OneDrive/Escritorio/Laboratorios Paradigmas/Lab Prolog/codigo fuente/tdaOption_21077657_CernaVargas.pl').
consult('C:/Users/Tamara/OneDrive/Escritorio/Laboratorios Paradigmas/Lab Prolog/codigo fuente/tdaFlow_21077657_CernaVargas.pl').
consult('C:/Users/Tamara/OneDrive/Escritorio/Laboratorios Paradigmas/Lab Prolog/codigo fuente/tdaChatbot_21077657_CernaVargas.pl').
consult('C:/Users/Tamara/OneDrive/Escritorio/Laboratorios Paradigmas/Lab Prolog/codigo fuente/tdaSystem_21077657_CernaVargas.pl').
consult('C:/Users/Tamara/OneDrive/Escritorio/Laboratorios Paradigmas/Lab Prolog/codigo fuente/tdaUser_21077657_CernaVargas.pl').
consult('C:/Users/Tamara/OneDrive/Escritorio/Laboratorios Paradigmas/Lab Prolog/codigo fuente/tdaChatHistory_21077657_CernaVargas.pl').
true.

?- true.

?- true.

?- Warning: c:/users/tamara/onedrive/escritorio/laboratorios paradigmas/lab prolog/codigo fuente/tdasystem_21077657_cernavargas.pl:108:
Warning:    Singleton variables: [User1]
true.

?- Warning: c:/users/tamara/onedrive/escritorio/laboratorios paradigmas/lab prolog/codigo fuente/tdauser_21077657_cernavargas.pl:6:
Warning:    Singleton variables: [NewChat,NewUser,Usuario]
Warning: c:/users/tamara/onedrive/escritorio/laboratorios paradigmas/lab prolog/codigo fuente/tdauser_21077657_cernavargas.pl:8:
Warning:    Singleton variables: [Usuario,NewChat,NewUser]
true.

?- true.

?-
```

Figura 1: llamado a consult

```
?- option(1, "1) Viajar", 1, 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),
chatbot(0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1, [F11], CB0),

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),
flow(1, "Flujo 1 Chatbot1\n¿Dónde te Gustaría ir?", [OP3,OP4,OP5,OP6], F20),

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),
flow(2, "Flujo 2 Chatbot1\n¿Qué atractivos te gustaría visitar?", [OP7,OP8,OP9,OP10], F21),

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(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),

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),
systemAddChatbot(S0, CB1, S01),
systemAddChatbot(S01, CB2, S02).
```

Figura 2: pegar consultas en la consola

```
OP7 = [1, "1) Central Park", 1, 2, ["Central", "Park", "Central Park"]],
OP8 = [2, "2) Museos", 1, 2, ["Museo"]],
OP9 = [3, "3) Ningún otro atractivo", 1, 3, ["Museo"]],
OP10 = [4, "4) Cambiar destino", 1, 1, ["Cambiar", "Volver", "Salir"]],
F21 = [2, "Flujo 2 Chatbot1\n¿Qué atractivos te gustaría visitar?", [[1, "1) Central Park", 1, 2, [...|...]], [2, "2) Museos",
 1, 2|...], [3, "3) Ningún otro atractivo", 1|...], [4, "4) Cambiar destino"|...]]],
OP11 = [1, "1) Solo", 1, 3, ["Solo"]],
OP12 = [2, "2) En pareja", 1, 3, ["Pareja"]],
OP13 = [3, "3) En familia", 1, 3, ["Familia"]],
OP14 = [4, "4) Agregar más atractivos", 1, 2, ["Volver", "Atractivos"]],
OP15 = [5, "5) En realidad quiero otro destino", 1, 1, ["Cambiar destino"]],
F22 = [3, "Flujo 3 Chatbot1\n¿Vas solo o acompañado?", [[1, "1) Solo", 1, 3, [...]], [2, "2) En pareja", 1, 3|...], [3, "3) En
 familia", 1|...], [4, "4) Agregar más atractivos"|...], [5|...]]],
CB1 = [1, "Agencia Viajes", "Bienvenido\n¿Dónde quieres viajar?", 1, [[1, "Flujo 1 Chatbot1\n¿Dónde te Gustaría ir?", [...|...
]], [2, "Flujo 2 Chatbot1\n¿Qué atractivos te gustaría visitar?"|...], [3|...]]],
OP16 = [1, "1) Carrera Técnica", 2, 1, ["Técnica"]],
OP17 = [2, "2) Postgrado", 2, 1, ["Doctorado", "Magister", "Postgrado"]],
OP18 = [3, "3) Volver", 0, 1, ["Volver", "Salir", "Regresar"]],
F30 = [1, "Flujo 1 Chatbot2\n¿Qué te gustaría estudiar?", [[1, "1) Carrera Técnica", 2, 1, [...]], [2, "2) Postgrado", 2, 1|..
.], [3, "3) Volver", 0|...]]],
CB2 = [2, "Orientador Académico", "Bienvenido\n¿Qué te gustaría estudiar?", 1, [[1, "Flujo 1 Chatbot2\n¿Qué te gustaría estudi
ar?", [...|...]]]],
S0 = [[], "Chatbots Paradigmas", 0, [[0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1|...]]],
S01 = [[], "Chatbots Paradigmas", 0, [[1, "Agencia Viajes", "Bienvenido\n¿Dónde quieres viajar?", 1|...], [0, "Inicial", "Bien
venido\n¿Qué te gustaría hacer?"|...]]],
S02 = [[], "Chatbots Paradigmas", 0, [[2, "Orientador Académico", "Bienvenido\n¿Qué te gustaría estudiar?", 1|...], [1, "Agenc
ia Viajes", "Bienvenido\n¿Dónde quieres viajar?"|...], [0, "Inicial"|...]]],
User1 = ["user1", []],
User2 = ["user2", []],
User3 = ["user3", []],
S2 = [[["user1", []]], "Chatbots Paradigmas", 0, [[2, "Orientador Académico", "Bienvenido\n¿Qué te gustaría estudiar?", 1|...]
, [1, "Agencia Viajes", "Bienvenido\n¿Dónde quieres viajar?"|...], [0, "Inicial"|...]]],
S3 = [[["user1", []], ["user2", []]], "Chatbots Paradigmas", 0, [[2, "Orientador Académico", "Bienvenido\n¿Qué te gustaría est
udiar?", 1|...], [1, "Agencia Viajes", "Bienvenido\n¿Dónde quieres viajar?"|...], [0, "Inicial"|...]]],
S5 = S9, S9 = [[["user1", []], ["user2", []], ["user3", []]], "Chatbots Paradigmas", 0, [[2, "Orientador Académico", "Bienveni
do\n¿Qué te gustaría estudiar?", 1|...], [1, "Agencia Viajes", "Bienvenido\n¿Dónde quieres viajar?"|...], [0, "Inicial"|...]]]

S7 = [[["user1", []], ["user1", []], ["user2", []], ["user3", []]], "Chatbots Paradigmas", 0, [[2, "Orientador Académico", "Bi
envenido\n¿Qué te gustaría estudiar?", 1|...], [1, "Agencia Viajes", "Bienvenido\n¿Dónde quieres viajar?"|...], [0, "Inicial"|
...]]],
S10 = [[["user2", []], ["user1", []], ["user2", []], ["user3", []]], "Chatbots Paradigmas", 0, [[2, "Orientador Académico", "B
ienvenido\n¿Qué te gustaría estudiar?", 1|...], [1, "Agencia Viajes", "Bienvenido\n¿Dónde quieres viajar?"|...], [0, "Inicial"
```

Figura 3: respuesta de la consola

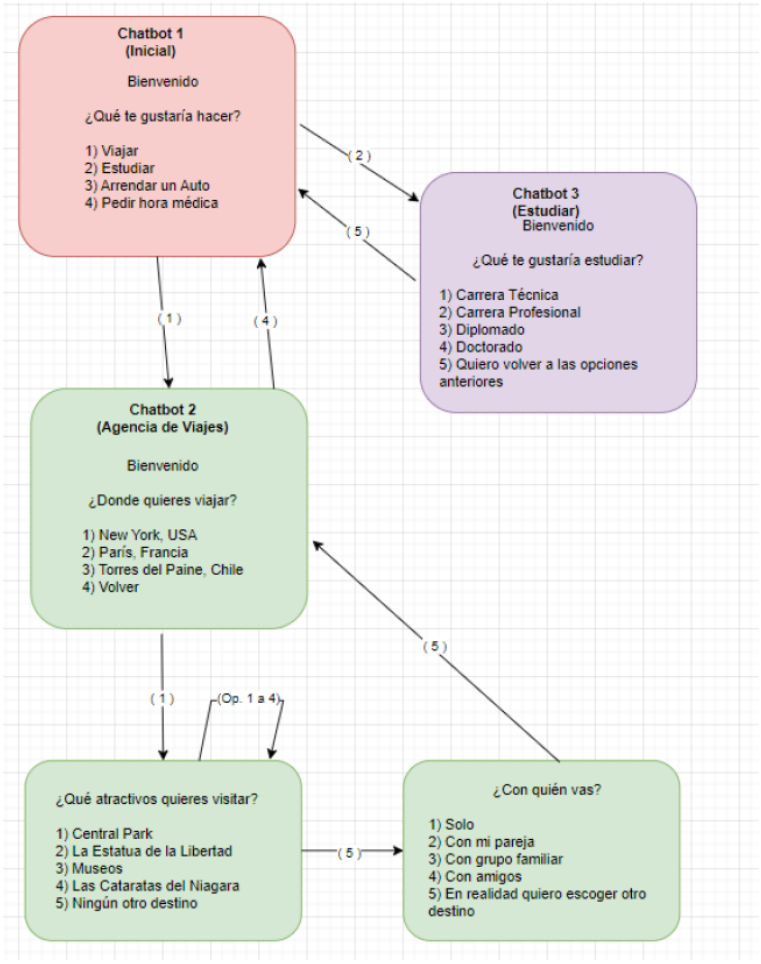

Figura 4: diagrama script de prueba de ejemplo

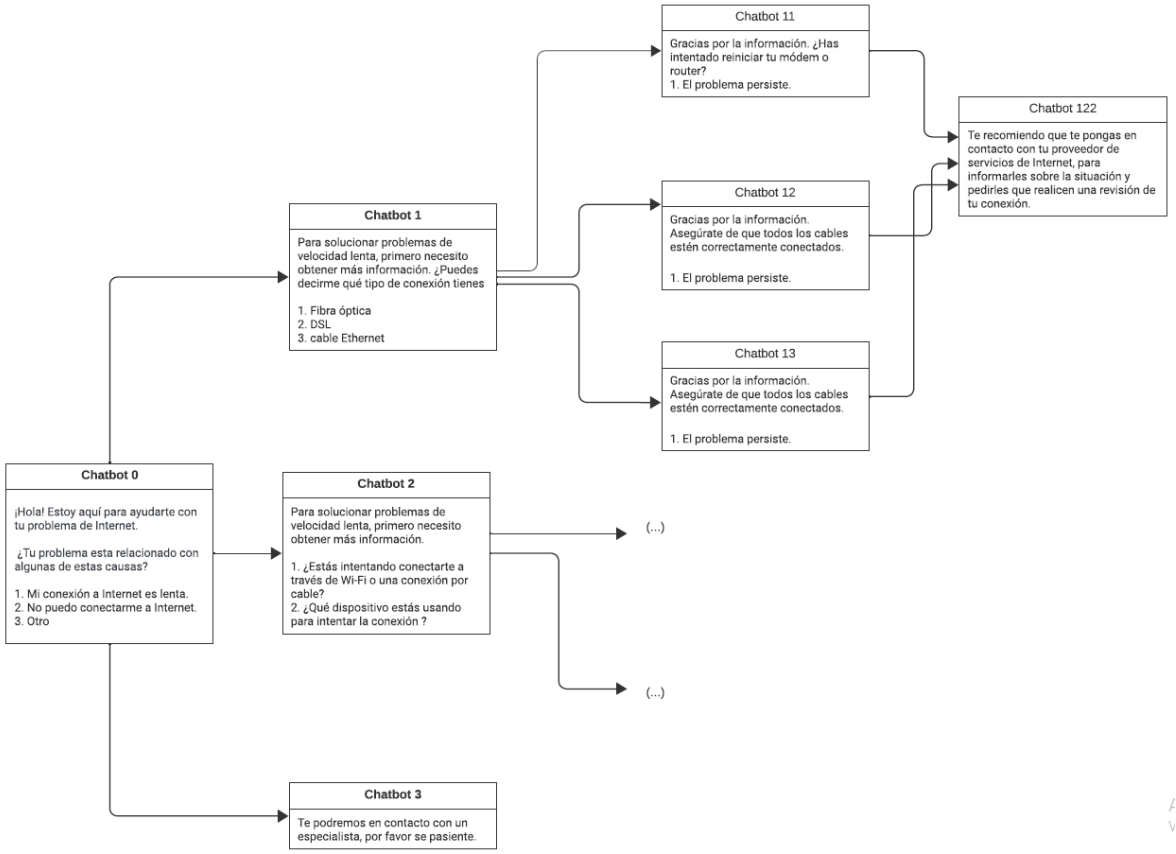

Figura 5: diagrama script de prueba 2

Link del código con implementaciones que no se pudieron lograr:

https://drive.google.com/file/d/1r_zJdbJ39s4E-5mTsfw6V5Nf79kZH_r8/view?usp=sharing