Informe Laboratorio 2
Paradigma Lógico

Estudiante: Benjamín Zúñiga J.

Asignatura: Paradigmas de Programación

Profesor: Roberto González I.

Noviembre 2023

Tabla de Contenido

Introducción

En la actualidad el masivo aumento del uso de las tecnologías digitales en la mayoría de los ámbitos de la vida cotidiana ha llevado a que el uso de un chatbot puede ser algo muy útil para cualquiera que quiera automatizar sus procesos, debido a que este una vez creado e implementado este en una web, aplicación, etc. Permite interactuar con usuarios a tiempo completo y entregar respuestas a estos de manera casi instantánea, a parte que tampoco significa un costo extra a quien esté buscando implementarlo para el uso particular que le quiera dar, todo esto ha provocado que en el último tiempo la "fama" e interés sobre los chatbots ha aumentado de gran manera. Ahora en lo que respecta a este informe, se presentaran distintos apartados tales como: una descripción del problema, una descripción del paradigma, un análisis del problema, diseño de la solución, aspectos de implementación, instrucciones de uso, resultados y conclusión, apartados que son importantes para entender cómo se llegó a la solución alcanzada del problema que se presenta para este trabajo, el cual es crear un sistema de chatbots a través del paradigma de programación lógico, el cual al ser parte del paradigma de programación declarativo se enfoca más en "Qué" resolver por sobre el "Cómo y más concretamente será una implementación por medio del lenguaje de programación Prolog que implementa el paradigma lógico de manera más "pura" que otros lenguajes que implementan este paradigma.

Descripción del problema

Para efectos de este trabajo tal como fue mencionado en el apartado anterior el problema que se busca solucionar es la creación de un sistema de chatbots ITR (Respuesta de Interacción a Texto) que permita a un usuario interactuar con este mediante texto y luego ser derivado a un chatbot especialista en el tema que quiera el usuario, a su vez cada uno de estos chatbots debe estar compuesto por flujos, los cuales van a permitir "desplazarse" por el sistema y al mismo tiempo estos flujos deben estar compuestos de la opciones que finalmente son las que permiten interactuar al usuario que pueda llegar a utilizar este sistema, en cuanto al "como" se debe tratar de lograr la creación de este sistema es por medio de una vista bajo el paradigma de programación lógica, lo cual tiene beneficios, como, que todo debe ser visto como una relación lógica, es decir solo tiene dos posibles resultados, verdadero o falso, pero también tiene desventajas y/o limitaciones, que van de la mano de le mencionado anteriormente ya que muchas veces puede ser una tarea compleja el abstraer un proceso a una conjunción de relaciones lógicas.

Descripción del paradigma

Tal como fue mencionado anteriormente el paradigma sobre el cual se va a basar la solución propuesta próximamente es en el paradigma lógico, el cual se basa principalmente en pensar, considerar todo como una relación, predicado o sentencia lógica, la cual puede ser verdadero o falso solamente, ya que esta es la unidad de abstracción básica del paradigma. Para entender más en profundidad este paradigma y como se trabaja bajo esta mirada sobre la

programación, es necesario entender conceptos como la lógica proposicional sobre la cual se construyen los predicados los cuales a su vez se pueden "dividir" en hechos, los cuales se entiende como un predicado que siempre es verdadero, y en reglas las cuales al consultar un predicado con un elemento concreto, su resultado (verdadero o falso) va a estar dado por el resultado de una conjunción de otros predicados mediante la lógica proposicional, ejemplos de conjunciones pueden ser el "y" u "o" lógico. También es necesario entender el concepto de backtraking, ya que este proceso es realizado de manera automática para consultar la base de conocimiento de cualquier programa creado bajo este paradigma, este proceso de backtracking o "vuelta atrás" se puede entender como un "proceso sistemático de búsqueda exhaustiva, es decir, enumerar todas las posibles configuraciones que satisfacen ciertos requisitos" (Villanueva, M. (2002), Algoritmos: Teorías y aplicaciones). Además es necesario comprender el concepto de unificación, ya que al uno consultar a la base de conocimiento también se podría pedir obtener un "valor" el cual haga que un predicado en específico sea verdadero, y es por medio de la unificación que se puede lograr encontrar ese "valor". Finalmente también es importante recordar y recalcar que la programación lógica es declarativa lo cual implica que el foco a la hora de solucionar un problema es el "qué" por sobre el "cómo", siendo en este caso que el objetivo es lograr plantear el problema en la base de conocimientos, debido a que gracias al backtraking automático y la unificación que se puede decir que el problema se resolverá casi solo.

Análisis del problema

Al profundizar en el problema, cabe mencionar que la creación de este sistema de chatbots tiene algunos requisitos y/o restricciones específicas principalmente relacionados a los dominios de cada predicado, es decir, el tipo de dato que ingresa al predicado y si necesitan utilizar recursión obligatoriamente algún predicado en particular, también el cómo se deben implementar los TDAs que representaran las "partes" que componen este sistema de chatbots, tales como los chatbots, flujos, opciones y el sistema, ya que a la hora de construir estos, aparte de las restricciones respecto al dominio de los predicados constructores tanto los chatbots, flujos y opciones manejan un id que sirve como verificador par evitar que alguna de estas estructuras esté repetida y evitar errores, además es necesario implementar predicados que permitan registrar usuarios en el sistema una única vez, es decir, no poder registrar dos veces el mismo usuario, luego que alguno de los usuarios registrados de pueda "loguear", que solo exista un usuario logueado a la vez, y solo si es que existe un usuario logueado pueda interactuar con el sistema, guardando su interacción en un historial propio de ese usuario, también implementar un predicado de "deslogueo" el cual permite quitar del estado logueado al usuario que lo este y por ultimo implementar un predicado que permita un simulación pseudoaleatoria de conversación con el sistema, donde el máximo de interacciones del sistema esté dado por un número que el usuario entregue.

Diseño de la solución

Respecto al diseño de solución, es importante comenzar mencionando que el principal enfoque fue la funcionalidad de tanto los TDAs como los propios predicados, es decir, lo principal fue lograr que cada predicado y TDA cumpliera su objetivo “como fuera” por sobre otros aspectos, siempre respetando los requisitos específicos de cada predicado y TDA.

En relación a la descomposición del problema lo primero realizado fue intentar graficar un modelo del sistema para comprenderlo visualmente y entender en qué lugar del modelo va cada estructura y cuales su función, cada estructura fue representada mediante un TDA distinto, en este caso los diseñados y empleados fueron el TDA Option, Flow, Chatbot, System, Chathistory, siendo tanto Option como Chathistory los TDAs más “simples” se podría decir, debido a que están compuestos netamente de estructuras y tipos de datos nativos del lenguaje, mientras que Flow, chatbot y system están compuestos de otros TDAs, como se puede ver en la imagen 1 en la sección “Anexos”.

Respecto a algoritmos o técnicas usados para alunas solucionar problemas particulares, no se usaron ningunas en particular más que recursión en caso de estar indicado por los requerimientos específicos, y a la hora de diseñar los constructores de cada TDA, debido a la necesidad de verificar que tanto en el TDA Flow, Chatbot y System era necesario verificar que cada “sub-tda” tuviera un id único en cada “contenedor de estos, por lo tanto la opción en la cual fue más fácil lograr esta tarea fue diseñar predicados recursivos.

Por último lo que fue de más ayuda durante el diseño de la solución a la hora de implementarla fue revisar la mayoría de predicados que ofrece el lenguaje Prolog sobre el cual está implementada está solución , en especial las que trabajan con listas, ya que esta fue la estructura básica para la implementación de cada TDA, aparte de diseñar una amplia variedad de predicados que cumplieran con objetivos más específicos en vez de intentar conjuntar varios predicados nativos del lenguaje, así disminuyendo probabilidades de fallo.

Aspectos de implementación

Respecto a la implementación esta fue desarrollada en el ide Swi-Prolog en su versión 9.0.4, el cual es para programar en el lenguaje Prolog, en el desarrollo de este no fue usada ninguna biblioteca externa, solo predicados nativos del lenguaje y la estructura de la implementación del proyecto es 6 archivos con los predicados extra, no obligatorios, correspondientes a cada TDA, 1 archivo principal llamado “main” el cual contiene la implementación de las funciones obligatorias, cada uno de estos archivos contiene la respectiva documentación de cada predicado dentro de estos, y por último un archivo de pruebas el cual contiene un script para probar la implementación contextualizada, con comentarios de cómo debería funcionar cada predicado.

Instrucciones de uso

Relacionado a las instrucciones de uso es importante dejar en claro que se asumió que el usuario va a ingresar cosas coherentes y respetar los dominios de cada predicado a la hora de consultar la base de conocimiento, es decir, por ejemplo en flows de un chatbot se van a ingresar efectivamente "n" Flow y no options, o que a la hora de buscar mensajes cuando se conversa con el sistema se ingresan mensajes relacionados o los números de las opciones son correlativos y siempre comienzan en 1, al igual que los ids de los TDAs, la consulta puede ser realizada consultando tanto el archivo pruebas como el archivo main. Luego de mencionar lo anterior se espera como resultado que el usuario pueda usar el cien por ciento de las funcionalidades obligatorias, sin errores siempre que siga lo mencionado anteriormente. (Ejemplo de una consulta en imagen 2 en la sección "Anexos")

Resultados y Autoevaluación

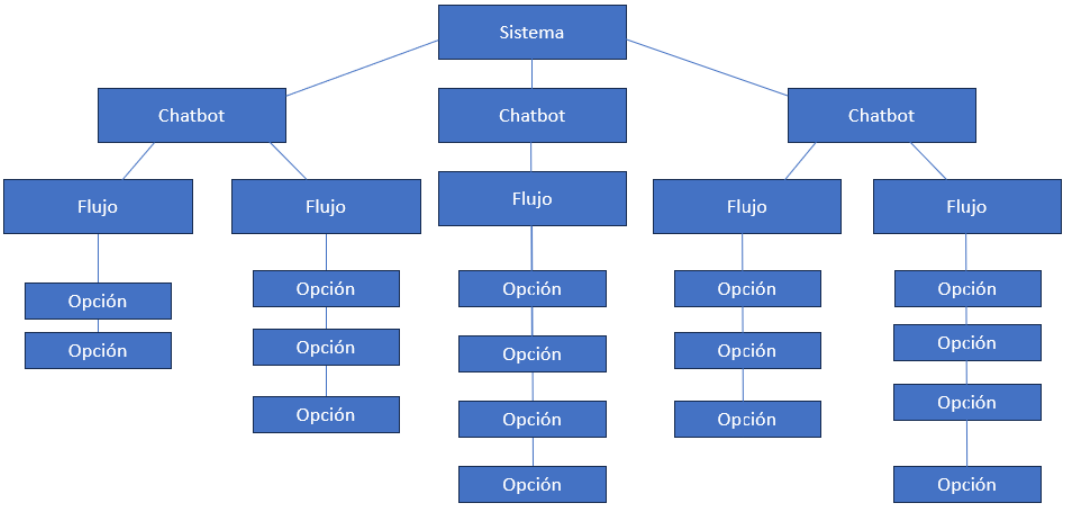
Los resultados alcanzados fueron bastante satisfactorios, ya que se logró implementar todos los predicados obligatorios de buena manera, sin ninguna dificultad mayor, considerando el grado de dificultad inherente a cada predicado y cumpliendo los requisitos específicos para cada una de estos, aparte se logró implementar predicados no obligatorios como predicados de pertenencia que permitieron poner en práctica tanto el paradigma como los conceptos aprendidos más en profundidad y a su vez facilitaron la implementación de otros predicados, ya que cada predicado no carga con una gran responsabilidad de cumplir varias metas secundarias. El proceso de prueba para obtener los resultados mencionados anteriormente fueron consultar a la base de conocimiento, compuesta por los 6 archivos mencionados anteriormente, el script de prueba dado en el enunciado de este trabajo, aparte del propio el cual se encuentra comentado y disponible en el archivo pruebas, se puede observar el resultado del script de prueba en la "Imagen 3" disponible en la sección "Anexos" de este trabajo. Por último existe una tabla de autoevaluación disponible en la sección "Anexos" como "Tabla 1" indicando el puntaje que se cree haber obtenido en cada requerimiento funcional de este trabajo.

Conclusión

Finalmente a modo de conclusión se puede decir que se logró solucionar el problema de buena manera, como se mencionaba en el apartado "Resultados y Autoevaluación", es importante considerar que el alcance la solución no es muy grande, debido a que se simula de buena manera y funciona, pero de manera relativamente primitiva, predicados, los cuales por parte de los requisitos de implementación que existen no muestran a tiempo real las propias interacciones, limitando el hecho de que esta implementación pudiera ser utilizada en un contexto de la vida real, aparte de que el tiempo que podría llegar a demorar en reaccionar el programa puede ser considerable considerando que su funcionamiento se basa en backtracking, por otra parte ya comparando la implementación realizada mediante el paradigma funcional, cabe decir que esta fue un poco más fácil debido a que una vez uno

expresa un predicado los resultados son mucho más limitados que los que puede arrojar una función, aparte de que la abstracción realizada en el trabajo anterior cumplía en su mayoría con los nuevos requisitos y siendo bastante eficaz a la hora de volver a implementar las funcionalidades en este nuevo paradigma, teniendo sí que modificar algunas cosas, como separar los usuarios logueados de registrados, lo que sí es importante recalcar que en un principio no fue una tarea fácil salir del paradigma funcional, tanto como lograr expresar las antiguas funciones ahora como predicado y que cada una entregue un verdadero o falso o mediante la unificación entregue el "valor" deseado.

Anexos



(Imagen 1: Representación del sistema)

```
?- 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), %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),
```

(Imagen 2: Consulta del script de prueba del enunciado)

```
1699888250.70309 - user2: hola
1699888250.703101-Inicial: flujo1
1) Viajar
2) Estudiar
1699888250.703115 - user2: 1
1699888250.703121-Agencia Viajes: Flujo 1 Chatbot1
¿Dónde te Gustaría ir?
1) New York, USA
2) París, Francia
3) Torres del Paine, Chile
4) Volver
1699888250.703144 - user2: 1
1699888250.70315-Agencia Viajes: Flujo 2 Chatbot1
¿Qué atractivos te gustaría visitar?
1) Central Park
2) Museos
3) Ningún otro atractivo
4) Cambiar destino
1699888250.703173 - user2: museo
1699888250.703179-Agencia Viajes: Flujo 2 Chatbot1
¿Qué atractivos te gustaría visitar?
1) Central Park
2) Museos
3) Ningún otro atractivo
4) Cambiar destino
1699888250.703194 - user2: 1
1699888250.7032-Agencia Viajes: Flujo 2 Chatbot1
¿Qué atractivos te gustaría visitar?
1) Central Park
2) Museos
3) Ningún otro atractivo
4) Cambiar destino
1699888250.703217 - user2: 3
1699888250.703223-Agencia Viajes: Flujo 3 Chatbot1
¿Vas solo o acompañado?
1) Solo
2) En pareja
3) En familia
4) Agregar más atractivos
5) En realidad quiero otro destino
1699888250.703243 - user2: 5
```

(Imagen 3: String obtenido después de probar el script de prueba del enunciado)

N° Requerimiento funcional	Puntaje
1	1
2	1
3	1
4	1
5	1
6	1

7	1
8	1
9	1
10	1
11	1
12	1
13	1
14	1

(Tabla 1: Puntajes de autoevaluación)

Referencias

- Villanueva, M. (2002), Algoritmos: Teorías y aplicaciones.
- Keller B. Lecture 8: Logic Programming Languages. Extraido de: https://courses.cs.vt.edu/~cs3304/Spring02/lectures/lect08.pdf
- Toro, R. (2015), Backtracking. Extraido de: http://www.cs.toronto.edu/~rntoro/intro/21/C21.pdf
- Alfonso, L; Rivera, S y Valderrama, P. (2020), Programación Lógica. Extraido de: https://ferestrepoca.github.io/paradigmas-de-programacion/proglogica/logica_teoria/docs/2020-1.pdf