Universidad de Santiago de Chile
Facultad de Ingenieria
Departamento de Ingenieria Informatica

Laboratorio N°2

Sistema de chatbots en base al paradigma de programación logico

Rodrigo González García

Profesor: Roberto Gonzalez Ibáñez

Ramo: Paradigmas de la programación

Sección: 13310-0

Tabla de Contenidos.

1. Introducción

En este informe se abordará el proceso de desarrollo del segundo laboratorio de paradigmas de la programación, el cual consiste crear un sistema de gestión de chatbots de ITR (Respuesta de Interacción de texto).
Se comenzará con la descripción del problema a resolver, una descripción del paradigma a utilizar (En este caso siendo Prolog), seguido del análisis del problema, indagando en la representación de los elementos del sistema y como se trabajaran con estos, posteriormente se analizara el diseño de la solución, presentando el enfoque que se tomó para el desarrollo del problema y como se crearon los predicados que componen el programa, se continuará con los resultados obtenidos, junto con una autoevaluación finalizando con las conclusiones de este laboratorio

2. Descripción del Problema.

La problemática por afrontar es la creación de un sistema de creación y gestión de chatbots de diversos temas.
El sistema debe tener la capacidad de crear y almacenar varios chatbots, los cuales dentro de ellos deben contener flujos que representan el tema y la dirección que toma la interacción entre el usuario y el sistema, estos flow a su vez deben poder almacenar distintas opciones, las cuales permiten, a la hora de seleccionarlas, el redireccionamiento a otro flujo o chatbot, permitiendo cambiar de tema (como por ejemplo, un sistema que representa un centro comercial, donde los distintos chatbots representen tiendas y los flujos los distintos productos de la tienda).

3. Descripción del Paradigma.

El paradigma lógico consiste en una colección de hechos y reglas que describen las relaciones y propiedades de entidades (Linkedin, 2023), en otras palabras en la programación lógica se basa en la creación de una base de conocimiento, la cual posee hechos y reglas, las cuales se pueden consultar y el programa buscara una respuesta en utilizando la base anteriormente mencionada (Geeks for geeks, 2022), como el paradigma anterior, este también se enfoca en especificar el problema a solucionar y no el paso a paso de cómo se llega a la solución de este.
Los hechos que se declaran simbolizan relaciones u objetos, los cuales, si se consultan, darán sí o sí verdadero.
Las reglas representan relaciones entre términos, su veracidad no está asegurada y se componen de un consecuente y un antecedente, se comprueba la veracidad del consecuente al confirmarse la veracidad de todos los antecedentes, vale la pena destacar que se puede utilizar una regla dentro de sí misma, de esta manera creando una regla recursiva (Logic Programming, 2023).
A la hora de hacer una consulta, el programa busca una respuesta que dé como resultado true, esto se logra a través del uso de back tracking, ósea algoritmos que buscan todas las posibles combinaciones de elementos recursivamente con el objetivo de obtener todas las soluciones posibles al problema (Geeks for geeks, 2023), además de utilizar la unificación, ósea, hacer que dos elementos sean equivalentes.

4. *Análisis del Problema*

Para comenzar el análisis, vale la pena ver cuáles componentes dan forma al sistema, estos siendo, en orden de mayor a menor nivel: System, Chatbot, Flow, Option, con User y su historial siendo elementos relacionados con el sistema, pero se trabajará con ellos de manera más independiente.
Se sabe que el system debe poseer un nombre, el ID del chatbot inicial y una lista de chatbots, a su vez un chatbot está compuesto por su id, el cual debe ser único dentro del sistema, su nombre, mensaje de bienvenida, el ID de su flow inicial y su lista de flows, estos están compuestos por su ID (el cual también debe ser único dentro de su respectivo chatbot), su nombre y mensaje y una lista de sus respectivos option, para finalizar el análisis de la estructura del sistema los option se componen de un código, el mensaje de la opción, el ID del chatbot y del flow al cual se dirigirá la interacción del usuario y una lista de keywords característica de cada opción.
Teniendo en cuenta las características que presenta cada componente del sistema, se concluye que la forma en la que se representaran será en la forma de listas debido a que, al estandarizar la representación de cada elemento, se puede acceder fácilmente a cada uno de sus partes y además permite la creación de predicados universales que pueden ser usados en más de un elemento (como por ejemplo ver si hay elementos repetidos, ya sea por el elemento completo o por la ID que posea un elemento, lo cual se puede usar tanto en el flow como en los chatbots).
Debido a que el usuario debe interactuar con el sistema, uno de los elementos más importantes que se deben guardar son las ID del chatbot y el flow que están siendo usados actualmente, los cuales deben ser guardados en el sistema y en el chatbot que se están utilizando, pero esto crea un problema, a la hora de empezar desde cero una interacción (como debe suceder al momento de cerrar sesión con un usuario y abrir sesión con otro) también se debe tener a mano todos los valores iniciales del sistema, es por esto que se deberán guardar los ID que se vayan a cambiar a la hora de interactuar con el sistema.
También cada usuario debe tener vinculado un historial de su interacción con el sistema, lo cual se puede lograr de una manera práctica al tener en cuenta que cada usuario debe estar registrado en el sistema para poder interactuar con él, por esto, el usuario deberá tener un historial base a la hora de añadirse al sistema, lo cual se puede lograr generando un par que contenga el nombre del usuario y su historial, el cual debe ser actualizado al finalizar la interacción.

5. Diseño de la Solución

Para comenzar, se utilizará una técnica de diseño Bottom-up, la cual consiste en crear las partes independientes del sistema de manera detallada para luego enlazarlas, formando componentes más grandes (Wiki Adsi, s.f.).

Empezando por el TDA Option, se implementa la option como una lista de 5 elementos, un código utilizado a la hora de interactuar con el sistema, un mensaje, el ID del Chatbot y el ID del Flow al que se dirigirá el sistema después de seleccionar la opción y las keywords de esta, representadas como una lista de strings.

Además del constructor, solo se implementaron los selectores de los elementos del option.

En TDA Flow, se representa el flow con una lista de 3 elementos un ID, un mensaje y una lista de options, además de los selectores se implementa el predicado flowAddOption, el cual utiliza los selectores para obtener cada elemento del flow, los predicados appendfin y checkrepeatid, que serán explicados al final esta sección, para modificar la lista de options y asegurar que no hayan repetidos, para finalmente usar el constructor para devolver un nuevo flow modificado.

Continuando con el TDA Chatbot, cada chatbot se representa de una manera más especial, además de los elementos básicos en la lista, estos siendo un ID, el nombre del Chatbot, el ID del flow inicial (aunque se utiliza como el flow actual del chatbot) y una lista de flows (que puede ser vacia), se añade un elemento extra al final de esta, siendo el flow inicial del Chatbot, este no será modificado nunca y se utilizará a la hora de devolver el sistema a su estado inicial.

Además de los selectores, se añaden los predicados selectores, el modificador chatbotAddFlow, que funciona utilizando los selectores y añadiendo el flow nuevo al final de la lista de flows y los predicados rechatbot y chatbotresetid, permitiendo recrear un chatbot totalmente, obteniendo todos sus elementos, y permitiendo volver a colocar el flow actual como flow inicial, respectivamente

Subiendo al último nivel de la estructura, al TDA System, el sistema se representa como una lista que contiene un nombre, el ID del Chatbot actual, el usuario actual, la lista de usuarios (cada usuario se compone por el nombre de usuario y su respectivo historial), una lista de Chatbots y el ID del Chatbot inicial (el cual será el mismo que el actual a la hora de crear el sistema).

Además de los selectores se implementan los predicados resystem, el cual recrea un system totalmente, permitiendo modificar cualquiera de sus elementos de manera independiente, los modificadores systemAddChatbot el cual añade un Chatbot siempre y cuando no este repetido, el systemAddUser que añade un usuario a la lista de usuarios, bajo la misma condición que el predicado anterior, el systemLogin, que inicia sesión del usuario siempre que esté en la lista de usuarios y el usuario actual sea un string vacío y el systemLogout, que permite cerrar la sesión del usuario actual, este predicado utiliza otro llamado systemresetchatbots, el cual reinicia los ID del flow actual de todos los Chatbots del sistema, además, a la hora de cerrar sesión también se reinicia el Chatbot actual.

Los TDA fuera de la estructura del sistema son el TDA User, que permite crear usuarios y añadir texto a su historial (createuser y useraddhistory respectivamente), y más importantemente el TDA Utilidades, el cual posee predicados de manejo de listas, entre los más importantes el memberid, que revisa que un ID no este presente dentro de una lista de elementos que posean ID, appendfin, que añade un elemento al final de una lista y checkrepeatid, el cual recibe solo una lista y revisa que ningún elemento posea el mismo ID que otro.

6. Aspectos de la Implementación

Además de los TDA adicionales, todos los predicados modificadores que detecten un elemento repetido retornaran false, anteriormente se devolvía la lista sin modificar, pero se cambió debido a los requerimientos del enunciado

7. Instrucciones de Uso

El programa se utiliza haciendo cadenas de consultas, dejando una variable libre como resultado, el cual se utiliza en la consulta siguiente, de esta manera creando la estructura del sistema de Chatbot, si la consulta es correcta y no se presenten elementos repetidos, se creará el elemento deseado.

Para hacer una cadena de consultas, se separa cada predicado con una coma y, al terminar la consulta, se añade un punto final

Para ejemplificar una consulta, se presenta una como ejemplo:

```
option(1,"Modelos de plastico",1,2,["gunpla","modelo","modelo de plastico"],Op1),
option(2,"Herramientas de modelaje",1,2,["Herramientas","Alicate","Lija"],Op2),
flow(2,"Bienvenido a la tienda de hobby",[Op1,Op2],F1),
option(3,"Pintura",1,2,["Pintura","Tamiya","GW","paint","pintar"],Op3),
flowAddOption(F1,Op3,F2)
```

Cuadro N°1: Consulta de ejemplo de creación de un Flow con 3 Opciones

Este ejemplo da como resultado lo siguiente:

```
F1 = [2, "Bienvenido a la tienda de hobby", [[1, "Modelos de plastico", 1, 2, ["gunpla", "modelo", "modelo de plastico"]], [2, "Herramientas de modelaje", 1, 2, ["Herramientas", "Alicate", "Lija"]]]],
F2 =
[2, "Bienvenido a la tienda de hobby",
[[1, "Modelos de plastico", 1, 2, ["gunpla", "modelo", "modelo de plastico"]], [2, "Herramientas de modelaje", 1, 2, ["Herramientas", "Alicate", "Lija"]], [3, "Pintura", 1, 2, ["Pintura", "Tamiya", "GW", "paint", "pintar"]]]
,
Op1 = [1, "Modelos de plastico", 1, 2, ["gunpla", "modelo", "modelo de plastico"]],
Op2 = [2, "Herramientas de modelaje", 1, 2, ["Herramientas", "Alicate", "Lija"]],
Op3 = [3, "Pintura", 1, 2, ["Pintura", "Tamiya", "GW", "paint", "pintar"]]
```

Figura N°1: Resultado de consulta de ejemplo

Después de ejecutar una consulta, se pueden ejecutar cuantas consultas uno quiera

8. Resultados y Autoevaluación

Como resultado, no se alcanzaron a implementar la totalidad de los predicados funcionales requeridos, pero los que si fueron implementados funcionan correctamente y fueron modificados para cumplir con el requisito de que al fallar o encontrar elementos repetidos, los predicados debían devolver false

9. Conclusiones

P

Bibliografía

- *Difference Between Functional and Logical Programming*. (24 de febrero de 2022). Obtenido de Geeks for geeks: https://www.geeksforgeeks.org/difference-between-functional-and-logical-programming/
- *Logic Programming*. (8 de noviembre de 2023). Obtenido de Wikipedia contributors: https://en.wikipedia.org/wiki/Logic_programming
- *What are the main characteristics and examples of logic programming paradigms?* (12 de 6 de 2023). Obtenido de Linkedin: https://www.linkedin.com/advice/0/what-main-characteristics-examples-logic-programming
- *What are the main characteristics and examples of logic programming paradigms?* (12 de Junio de 2023). Obtenido de Linkedin: https://www.linkedin.com/advice/0/what-main-characteristics-examples-logic-programming

10. Anexos