UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

Departamento de Ingeniería Informática

Informe De Laboratorio: Creación De Un Sistema De Chatbots

Laboratorio 2 Paradigmas de la Programación

Gonzalo Moncada

21321047-5

Fecha:

13 de noviembre 2023

Profesor:

Gonzalo Martínez

TABLA DE CONTENIDO

1. CONTENIDO DEL INFORME

1.1. INTRODUCCIÓN

El informe presentado corresponde a la segunda entrega del laboratorio de la asignatura Paradigmas de Programación, asociado al paradigma lógico aplicado al lenguaje de programación Prolog. Los tópicos para cubrir en el mismo serán: Descripción del problema, descripción del paradigma, análisis del problema, diseño de la solución, aspectos de implementación, instrucciones de uso, resultados y autoevaluación y conclusiones. En este semestre el proyecto asignado es un sistema de chatbots.

1.2. DESCRIPCIÓN DEL PROBLEMA

Un sistema de chatbots permite a un usuario generar interacciones a través de un chat en base a un tópico especifico, es utilizado comúnmente para consultas y agendamientos o reservas. Un sistema de chatbots permite crear, vincular, interactuar y modificar las opciones, flujos y chatbots para realizar asi interacciones entre estos mismos, además de esto permite obtener una síntesis de lo realizado dentro del sistema.
Este proyecto se concentrará en trabajar en la generación de los vínculos de opciones y en la simulación de las interacciones entre el usuario y los chatbots, por consiguiente, no se contempla la implementación de interacciones por consola con espera de respuesta.

1.3. DESCRIPCIÓN DEL PARADIGMA

El paradigma lógico asociado a Prolog es un enfoque de programación que se basa en la lógica formal y la inferencia lógica para resolver problemas. Prolog, que significa "programación en lógica", es un lenguaje de programación utilizado para implementar este paradigma.
En el paradigma lógico, se describe el conocimiento y las reglas del dominio mediante hechos. Un programa Prolog está compuesto por un conjunto de predicados que definen relaciones entre objetos. Estos predicados se definen en términos de reglas lógicas y pueden representar

cualquier tipo de información, desde relaciones matemáticas hasta reglas del mundo real. Además de esto está la existencia de los hechos, los cuales son información de la base de conocimiento para el programa. La ejecución de un programa Prolog implica la resolución de consultas lógicas. El intérprete de Prolog busca en la base de conocimientos las cláusulas que se unifican con la consulta y realiza relaciones lógicas para encontrar soluciones. Esto se logra mediante la unificación (Javatpoint), y la aplicación de las reglas. El paradigma lógico ofrece una forma declarativa de programación, en la que se describe lo que se quiere obtener en lugar de como obtenerlo. La programación en Prolog se basa en la resolución de problemas mediante la búsqueda exhaustiva de soluciones.

1.4. ANÁLISIS DEL PROBLEMA

En específico, las funciones a realizar por el programa son: option, flow, flowAddOption, chatbot, chatbotAddFlow, system, systemAddChatbot, systemAddUser, systemLogin, systemLogout, systemTalkRec, systemSynthesis y systemSimulate. Esto en sistensis seria generar predicados que generen cada una de las estructuras demostradas (option, flow, chatbot, system) y sus correspondientes modificadores, además de esto permitir dentro del sistema el almacenamiento de usuarios los cuales puedan ser logeados y deslogeados, además se busca la manera de generar una interacción con el sistema para luego esta ser guardada en una parte del mismo, esto para lograr generar un formato de salida para la fácil comprensión de las interacciones hechas dentro del sistema, por último se busca generar una simulación de este con elecciones pseudoaleatorias.

1.5. DISEÑO DE LA SOLUCIÓN

El enfoque de solución emplea el uso constante de modificadores dentro de un TDA principal que será el de system. este TDA posee los atributos de Nombre (string), Usuarios (list) , Usuario Logeado (list) , ChatHistory (list), InitialChatbotCodeLink (Int), ActualChatbotCodeLink (int), ActualFlowCodeLink (int) y chatbots (list), todos estos atributos serán explicados a lo largo de esta sección, por lo cual se podría explicar con el flujo que cada sistema tiene un chatbot que tiene un flow que contiene opciones, por lo cual se utilizó distintos TDA para definir cada estructura con sus propios predicados de creación y modificación, debido a que en el enunciado se presentaron los dominios de cada una de las estructuras presentadas se definieron estos predicados como listas de cada atributo del dominio presentado, pero se presentó el

implementar una solucion la cual descartara las ocurrencias repetidas de los elementos al ingreso de una estructura (exceptuando las opciones debido a que son la estructura base), por lo cual debido a que el procedimiento era el mismo pero con distintas estructuras se optó por definir un TDA commons el cual tuviera los predicados "comunes" entre cada TDA, De esta manera, se creó un predicado que recorre la lista y, al encontrar un elemento presente en el resto de la lista, activa otro predicado encargado de eliminar todas las instancias de dicho elemento. El resultado es una lista libre de la presencia de dicho elemento, esto se realizó con la id ’s de cada lista de elemento para luego utilizar un predicado que recorra las id’ s y vaya ingresando el elemento como tal a una nueva lista para esta utilizarse en el creador de la estructura en específico. Luego de esto se logró realizar todos los predicados solicitados hasta el de añadir usuario, el cual se vio en la necesidad de agregar el Usuarios presentado en el inicio de esta sección, por lo cual solo se añadió el nombre y a este se le hacia la función member para poder verificar su existencia antes de agregarlo, luego al presentarse la función de Login y Logout se optó por agregar el Logeado presentado previamente, pero este en función de lista, esto debido a que se podía revisar si existe o no un elemento en esta lista para definir si alguien estaba logeado o no en el sistema, por último la función systemTalkRec fue la más importante en desarrollar, por lo cual se optó por agregar en el dominio de System: ActualChatbotCodeLink (int) y ActualFlowCodeLink (int), estos serían valores en los cuales se basaría la búsqueda de las opciones posibles o las keywords, además de esto estos se irían actualizando para que la siguiente interacción fuera en base a estos, por último en el momento de un logout estos podrán resetearse para generar interacciones con otro usuario, en lo demás se usó de base este predicado para generar un formato de string e imprimirlo con write, y generar una simulación con valores pseudoaleatorios que fuera desde un numero sacando el resto de este y colocándolo como elección del usuario, los vínculos entre los TDA están en el anexo .

1.6. ASPECTOS DE IMPLEMENTACIÓN

El compilador utilizado en este proyecto fue de Prolog siendo la versión 9.0.4. El código se estructura por TDA, para facilitar la edición a la hora de agregar contenido a alguno de estos, utilizando módulos para importar y exportar información de estos a otros archivos y asi hacer utilización de estos. Para este laboratorio no se utilizó bibliotecas externas, solo funciones nativas de prolog.

1.7. INSTRUCCIONES DE USO

Se tiene un archivo pruebas que contiene varios ejemplos y se tiene en funcionamiento solo los TDA requeridos (option, flow, chatbot y system). En cambio, para el uso fuera de los ejemplos propuestos en este archivo se presenta el anexo 2, el cual presenta la manera de generar cada uno de los requerimientos para realizar una interacción en el sistema, en cuanto a esta interacción será posible realizarse siempre y cuando los vínculos tengan una dirección correcta y exista un usuario logeado, ya que si llamo a un Flow inexistente se terminará el programa. En cuanto a posibles errores, mientras se utilicen correctamente los predicados en base al dominio presentado no debería existir errores.

1.8. RESULTADOS Y AUTOEVALUACIÓN

En el amplio aspecto, el proyecto fue exitoso, pues las funciones requeridas se pudieron realizar en su totalidad. Debido a que este es un lenguaje basado en lógica, se pudo observar cómo entradas erróneas daban el resultado esperado (false). Además, el script de pruebas entregado funciona sin romperse cuando se ejecuta de manera completa en la consulta del compilador presentado.

La autoevaluación radica entre valores de 0 (no implementado) hasta 1 (implementado y sin errores) aumentando en una escala de 0.25 y se puede observar en el anexo 3. Se les asigna a todas las funciones 1 dado que al momento de realizar diversas pruebas estas trabajan correctamente y sin errores.

1.9. CONCLUSIONES

La creación de bases de conocimiento con Prolog y el paradigma lógico es realmente una manera muy distinta de programar. Incluso con Racket, lo que generaba el mayor cambio respecto a la manera “tradicional” de programar, era el uso de la sintaxis LISP, sin embargo, el funcionamiento del lenguaje aún se asemeja al comportamiento típico de un programa. Por el contrario, Prolog entrega una manera completamente distinta de programar, aunque su sintaxis es más parecida a lo común, su comportamiento es abismalmente distinto.

Las primeras interacciones con el lenguaje fueron lo más complejo, pero una vez se entiende correctamente el concepto de unificación y la manera de utilizar la recursión de cola, todo entra

en sentido. Se podría decir que la mayor limitación fue la utilización de backtracking, debido a que en ciertas pruebas se observó que a pesar de soltar falso el programa volvió a buscar una opción a pesar de estar en una capa completamente distinta.

Finalmente, se puede decir que este paradigma pudo ser abordado de la mejor manera, debido a que se logró generar un proceso bastante lineal con ciertas dificultades en funciones específicas pero que no terminaron de detener el progreso del propio proyecto.

1.10. BIBLIOGRAFÍA

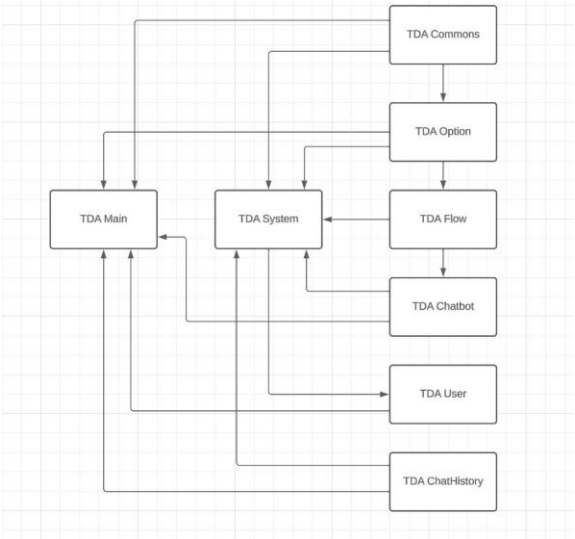
SWI-Prolog. (s. f.-b). SWI-Prolog. Recuperado 13 de noviembre de 2023, de https://www.swi-prolog.org/

Flores Sánchez, V. (2023, octubre). *Paradigmas de Programación Proyecto semestral de laboratorio Versión Preliminar - actualizada al 08/11/2023 Laboratorio 2 (Paradigma Lógico - Lenguaje Prolog)*. docs. Recuperado 13 de noviembre de 2023, de https://docs.google.com/document/d/1QCk3k-HCShNSByEZHoEIFbrxoOJSnPblyjiJDj3Q9Jk/edit

jecaestevez. (2010, 19 junio). *Prolog Proceso de unificacion*. wordpress. Recuperado 13 de noviembre de 2023, de https://programacionprolog.wordpress.com/2010/06/19/prolog-proceso-de-unificacion/

1.11. ANEXO

- Anexo 1: Flujo de los TDA en el programa



- Anexo 2: Instrucciones de Uso

```
Welcome to SWI-Prolog (threaded, 64 bits, version 9.0.4)
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).

?- option(1, "1 - viajar", 1, 1, ["viajar", "turistear", "conocer"], Option),
|    flow(1, "Flujo Principal Chatbot 1\nBienvenido\n¿Qué te gustaría hacer?", [Option], Flow),
|    chatbot(0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1, [Flow], Chatbot),
|    system("Chatbots Paradigmas", 0, [Chatbot], System).
Option = [1, "1 - viajar", 1, 1, ["viajar", "turistear", "conocer"]],
Flow = [1, "Flujo Principal Chatbot 1\nBienvenido\n¿Qué te gustaría hacer?", [[1, "1 - viajar", 1, 1, [...|...]]]],
Chatbot = [0, "Inicial", "Bienvenido\n¿Qué te gustaría hacer?", 1, [[1, "Flujo Principal Chatbot 1\nBienvenido\n¿Qué te gustaría hacer?", [...]]]],
System = ["Chatbots Paradigmas", [], [], [], 0, 0, 1, [[...|...]]].

?-
```

- Anexo 3: Autoevaluacion:

Autoevaluación	Puntaje
option	1
flow	1
flowAddOption	1
chatbot	1
chatbotAddFlow	1
system	1
systemAddChatbot	1
systemAddUser	1
systemLogin	1
systemLogout	1
systemTalkRec	1
systemSynthesis	1
systemSimulate	1