UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

INFORME DE LABORATORIO 2: CREACIÓN DE UN SISTEMA PARA IMPLEMENTAR CHATBOT EN PROLOG

Nombre: Alvaro Muñoz Araya

Profesor: Víctor Flores Sánchez

Asignatura: Paradigmas de Programación

Fecha: 13 de noviembre de 2023

Índice

1. Introducción 1
1.1 Descripción del problema 2
1.2 Descripción del paradigma 2
1.3 Objetivos 3
2. Desarrollo 3
2.1 Análisis del problema 3
2.2 Diseño de la solución 4
2.3 Aspectos de la implementación 6
2.4 Instrucciones de uso 6
2.5 Resultados y autoevaluación 7
3. Bibliografía y referencias 8

1. INTRODUCCIÓN

En este informe, se abordarán detalladamente las características del proyecto desarrollado en el marco de la asignatura de Paradigmas de la Programación, específicamente enfocado en la programación lógica. Mediante el uso del lenguaje de programación Prolog y el intérprete SWI Prolog, exploraremos a fondo las particularidades de este paradigma.

El documento iniciará con una exposición detallada de la situación planteada en el laboratorio, incluyendo una descripción del problema a resolver y una presentación del paradigma utilizado. A continuación, se establecerán los objetivos y se realizará un análisis exhaustivo del problema, dando paso a la propuesta de un diseño de solución.

La sección correspondiente a la implementación detallará el proceso seguido para llevar a cabo la solución propuesta, incluyendo las instrucciones de uso del programa desarrollado. Además, se presentarán los resultados obtenidos, seguidos de una autoevaluación del trabajo realizado.

Finalmente, el informe concluirá con una sección de conclusiones, resumiendo los hallazgos clave y destacando las lecciones aprendidas durante la realización de este proyecto en el contexto de la programación lógica.

1.1 DESCRIPCIÓN DEL PROBLEMA

En este laboratorio, se solicita la implementación de un Sistema de Chatbots utilizando el lenguaje Prolog. Este programa tiene como objetivo crear, vincular y permitir la interacción entre Chatbots, enfocándose en una interfaz de línea de comandos de manera estructurada. La interacción con el sistema se llevará a cabo mediante la introducción de palabras clave en la línea de comandos. Es importante destacar que dado las características del proyecto no se requiere la implementación de inteligencia artificial en este contexto.

1.2 DESCRIPCIÓN DEL PARADIGMA

Este primer laboratorio como ya antes se ha mencionado, se trabajará bajo el paradigma de programación logico. Este paradigma es un enfoque de desarrollo de software basado en la lógica matemática y la teoría de conjuntos.

La programación lógica es particularmente efectiva para problemas que involucran búsquedas, representación de conocimientos y procesamiento simbólico. Prolog es un ejemplo destacado de un lenguaje de programación lógica que se utiliza en una variedad de aplicaciones, como inteligencia artificial, procesamiento del lenguaje natural y sistemas expertos.

Algunos conceptos importantes a tener en cuenta de la programación logica son:

- Reglas y Hechos: En lugar de describir cómo se realiza un cálculo o una tarea, en programación lógica se especifican hechos y reglas lógicas que definen las relaciones entre los datos. Los hechos son afirmaciones que se consideran verdaderas, y las reglas establecen condiciones bajo las cuales se puede derivar una nueva afirmación.

- Backtracking: La programación lógica a menudo hace uso del backtracking. Si una regla o una rama de ejecución no conduce a una solución, el sistema vuelve atrás y explora otras posibilidades.
- Base de Conocimientos: Los hechos y reglas que definen las relaciones y el conocimiento en el programa se agrupan en una base de conocimientos. Esta base de conocimientos es consultada para realizar inferencias y responder preguntas.
- Supuesto del mundo cerrado: Este supuesto establece que todo lo que no está explícitamente afirmado como verdadero se considera falso. En otras palabras, la base de conocimientos se considera completa, y si no hay evidencia de un hecho, se asume que es falso.

1.3 OBJETIVOS

El principal objetivo es aprender el paradigma de la programación logica, como se implementa en una situación real, para que situaciones nos puede ayudar, además de sus ventajas y desventajas.
Otro objetivo es aplicar los conocimientos aprendidos del lenguaje Prolog, que provee de las herramientas necesarias para construir el sistema de Chatbots.

2. DESARROLLO

2.1 ANALISIS DEL PROBLEMA

Teniendo en cuenta que los chatbots que serán implementados al sistema corresponden al chatbot del tipo ITR (Respuesta de Interacción a Texto), debemos construir un ambiente propicio para que el usuario pueda crear su propio sistema.

Para ello debemos aclarar cada concepto que se verá involucrado en el sistema:

Chatbot: Es el encargado de contener la información necesaria para generar una interacción con el usuario, se asocia a un ID para poder generar conexiones con otros chatbots.

Options: Son las opciones de texto que tiene un chatbot en respuesta a una interacción especifica con el usuario, cada “Option” debe tener un numero asociado que nos permitirá seleccionarlo dentro del las demás opciones que contiene un Chatbot

System: Un sistema contiene a todos los Chatbots en su conjunto, hay una variedad de predicados que se relacionan con el sistema, como por ejemplo crear Chatbots, eliminarlos o modificarlos.

User: El identificador del usuario que ingresa al sistema, cada sistema creado es único en relación a cada usuario

Login: Nos permitirá acceder al sistema con un determinado usuario

2.2 DISEÑO DE LA SOLUCIÓN

2.2.1 TDA Option:

option: Predicado constructor del TDA. Es una representación de lista con la finalidad de construcción de opciones para el flujo de un Chatbot. La meta primaria es: option, metas secundarias son: number, string, number, number

isOption: Predicado de pertenencia del TDA. Verifica si el argumento es Option. La meta primaria es: isOption, meta secundaria es: option

2.2.2 TDA Flow:

flow: Predicado constructor del TDA. Es una representación en base a lista de la construcción de un Flow. Meta primaria: Flow, metas secundarias: number, string.

flowAddOption: Predicado modificador del TDA. Permite añadir una opción a un Flow, Meta primaria: flowAddOption, metas secundarias: getFlowId, flowGetMensagge, flowGetOptions, concatenar, not_member.

flowGetOptions: Predicado selector del TDA. Permite obtener las opciones de un Flow, metas primarias: flowGetOptions, metas secundarias: isFlow.

flowGetMessage: Predicado selector del TDA. Permite obtener los mensajes de un Flow, metas primarias: flowGetMessage, metas secundarias: isFlow.

getFlowId: Predicado selector del TDA. Permite obtener el Id de un Flow, metas primarias: getFlowId, metas secundarias: isFlow.

isFlow: Predicado de pertenencia del TDA. Permite verificar si el argumento es Flow, metas primarias: isFlow, metas secundarias: flow.

2.2.3 TDA Chatbot:

chatbot: Predicado constructor del TDA. Permite crear un chatbot en formato de lista, metas primarias: chatbot, metas secundarias: number, string, string, number

chatbotAddFlow: Predicado modificador del TDA. Permite añadir un flow al chatbot, metas primarias: chatbotAddFlow, metas secundarias, getChatbotId, getChatbotName, getChatbotMessage, getChatbotStartFlowId, getChatbotFlows, concatenar.

isChatbot: Predicado de pertenencia del TDA. Permite saber si el argumento es chatbot, metas primarias: isChatbot, metas secundarias: chatbot.

getChatbotId: Predicado selector del TDA. Permite obtener el Id de un chatbot, metas primarias: getChatbotId, metas secundarias: isChatbot.

getChatbotName: Predicado selector del TDA. Permite obtener el nombre de un chatbot, metas primarias: getChatbotName, metas secundarias: isChatbot.

getChatbotMessage: Predicado selector del TDA. Permite obtener el mensaje de un chatbot, metas primarias: getChatbotMessage, metas secundarias: isChatbot.

getChatbotStartFlowId: Predicado selector del TDA. Permite obtener el Flow de inicio de un chatbot, metas primarias: getChatbotStartFlowId, metas secundarias: isChatbot.

getChatbotFlows: Predicado selector del TDA. Permite obtener los Flows de un chatbot, metas primarias: getChatbotFlows, metas secundarias: isChatbot.

2.2.4 TDA System

system: Predicado constructor del TDA. Crea un nuevo sistema en formato de lista, metas primarias: system, metas secundarias: string, number, no_repetidos.

systemAddChatbot: Predicado modificador del TDA. Añade un chatbot al sistema, metas primarias: systemAddChatbot, metas secundarias: getSystemName, getSystemIntialChatbotCL, getSystemChatbotList, getSystemUser, getSystemChatHistory, getSystemUsersList, not_member, concatenar.

systemAddUser: Predicado modificador del TDA. Añade un usuario al sistema, metas primarias: systemAddUser, metas secundarias: getSystemName, getSystemIntialChatbotCL, getSystemChatbotList, getSystemUser, getSystemChatHistory, getSystemUsersList, concatenar.

systemLogin: Predicado modificador del TDA. Añade un usuario al apartado de usuarios logeados del sistema, metas primarias: systemLogin, metas secundarias: getSystemName, getSystemIntialChatbotCL, getSystemChatbotList, getSystemUser, getSystemChatHistory, getSystemUsersList, not_member, concatenar, isSystem.

systemLogout: Predicado modificador del TDA. Quita un usuario del apartado de usuarios logeados del sistema, metas primarias: systemLogout, metas secundarias: getSystemName, getSystemIntialChatbotCL, getSystemChatbotList, getSystemUser, getSystemChatHistory, getSystemUsersList, length, isSystem.

isSystem: Predicado de pertenencia del TDA. Permite saber si el argumento es un sistema, metas primarias: isSystem, metas secundarias, system, isUser, is_list, is_list.

isUser: Predicado de pertenencia del TDA. Permite saber si el argumento es un usuario, metas primarias: isUser, metas secundarias: length

getSystemName: Predicado selector del TDA. Permite obtener el nombre de un sistema, metas primarias: getSystemName, metas secundarias: isSystem.

getSystemInitialChatbotCL: Predicado selector del TDA. Permite obtener el código inicial de un chatbot dentro de un sistema, metas primarias: getSystemInitialChatbotCL, metas secundarias: isSystem

getSystemChatbotList: Predicado selector del TDA. Permite obtener la lista de chatbots dentro de un sistema, metas primarias: getSystemChatbotList, metas secundarias: isSystem

getSystemUser: Predicado selector del TDA. Permite obtener el usuario logeado de un sistema, metas primarias: getSystemUser, metas secundarias: isSystem

getSystemChatHistory: Predicado selector del TDA. Permite obtener el historial del chat de un sistema, metas primarias: getSystemChatHistory, metas secundarias: isSystem

getSystemUsersList: Predicado selector del TDA. Permite obtener la lista de usuarios creados en un sistema, metas primarias: getSystemUsersList, metas secundarias: isSystem

2.3 ASPECTOS DE IMPLEMENTACIÓN

Se ha usado el intérprete de Prolog: SWI Prolog en su versión 9.0.4.

Utilizando :-module y :-use_module se ha podido dividir el código en diferentes archivos, cada uno para un TDA distinto y para algunos archivos que contienen scripts de prueba. Los archivos creados son los siguientes: “TDAoption_207242381_MunozAraya.pl”, “TDAflow_207242381_MunozAraya.pl”, “TDAchatbot_207242381_MunozAraya.pl”, “TDAsystem_207242381_MunozAraya.pl”, “main.pl”, “pruebas_207242381_MunozAraya.pl”

No se han ocupado bibliotecas externas a las básicas incluidas en Prolog.

2.4 INSTRUCCIONES DE USO

Para poder ejecutar el código correctamente debemos tener todos los archivos ubicados en el mismo directorio. Seguido debemos abrir el intérprete de Prolog a disposición y consultar el archivo de script de pruebas “pruebas_207242381_MunozAraya.pl”, una vez consultado el archivo debemos escribir “test.” En el intérprete y automáticamente este cargará todas las consultas, cabe recalcar que al hacer uso de “write()” en el archivo de scripts, las variables se imprimirán en su totalidad en el intérprete, por lo que su lectura será dificultosa si se han creado muchas variables, sin embargo también podemos consultar el archivo “main.pl” copiando y pegando los scripts de prueba en la propia consulta

2.4.1 EJEMPLOS DE USO

En el siguiente ejemplo se muestra el archivo de pruebas, donde se puede ver el funcionamiento de todos los scripts

Se carga el archivo de pruebas y se escribe “test.” En el interprete.

También podemos consultar el archivo “main.pl” copiando y pegando los scripts a consultar

2.5 RESULTADOS Y AUTOEVALUACIÓN

Cabe mencionar que no todos los predicados fueron implementados exitosamente, dado los prerrequisitos de implementación, los predicados systemTalkRec, systemSynthesis, systemSimulate no funcionan para el 100% de los casos y necesitan correcciones.

Para el resto de los predicados, el sistema no presenta fallos conocidos.

2.5.1 AUTOEVALUACIÓN

Requerimientos Funcionales	Evaluación
TDAs	0.75
option	1
flow	1
flow-add-option	1
chatbot	1
chatbot-add-flow	1
system	1
system-add-chatbot	1
system-add-user	1
system-login	1
system-logout	1
system-talk-rec	0.25
system-synthesis	0
system-simulate	0

Nota esperada: 4.6

3 CONCLUSIÓN

Para concluir, cabe mencionar que el diseño de solución cumple en ciertos aspectos con el problema planteado inicialmente, Sin embargo, los predicados que permiten interactuar no fueron implementadas de manera exitosa, por lo que es difícil realizar una medición completa del sistema.
Por otra parte, el paradigma de programación lógico fue utilizado durante todo el proceso del proyecto, no se utilizaron otros conceptos fuera de lo que es el paradigma y se cumplió con los requerimientos mencionados a la hora de programar bajo este paradigma.

4 BIBLIOGRAFÍA Y REFERENCIAS

Paradigma logico uvirtual usach:
https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=17

Introduction to Prolog Programming (CSC240):
https://youtube.com/playlist?list=PLm8dSOaqLPHLMJbFJ4HbEVMsj9o8wfQ1j&si=-7jtbpgb_Oif-sBC