UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

Departamento de Ingeniería Informática

INFORME LABORATORIO 2 PARADIGMAS DE PROGRAMACIÓN

Aylin Castillo

Fecha:

13 de Noviembre 2023

TABLA DE CONTENIDO

1. CONTENIDO DEL INFORME

1.1. INTRODUCCIÓN

El informe presentado corresponde a la primera entrega del laboratorio de la asignatura Paradigmas de Programación, donde se tiene como objetivo dominar el paradigma lógico aplicándolo al lenguaje de programación Prolog. Los temas a 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 presentado consiste en un sistema de chatbots ITR (Respuesta de Interacción a Texto) donde usuarios pueden interactuar con ellos mediante opciones.

1.2. DESCRIPCIÓN DEL PROBLEMA

Un chatbot ITR tiene como objetivo responder a preguntas con opciones específicas. Estas opciones pueden estar dadas por palabras clave con sinónimos o con números o letras que permitan a un usuario escoger entre un listado de preguntas, donde cada una de ellas representa a un chatbot distinto.

Estos chatbots se comunicarán entre sí mediante flows (flujos), los cuales estarán compuestos por las opciones escogidas y números que los identificarán, para así evitar la repetición de estos.

Lo anteriormente mencionado se encontrará conectado en un sistema de chatbots, donde el usuario podrá iniciarlo e interactuar con las distintas opciones que ofrecerá este sistema, también podrá agregar preguntas y opciones, además podrá especificar enlaces para el flujo de interacción dentro del chatbot y hacia otros chatbots.

1.3. DESCRIPCIÓN DEL PARADIGMA

En este laboratorio se tiene como objetivo desarrollar el paradigma lógico en el lenguaje de programación Prolog. Este paradigma tiene como principal característica su enfoque basado en

la lógica formal y la inferencia lógica para resolver problemas, el cual se aplica mediante la creación de una “base de conocimiento” que contendrá los hechos del problema que se desea abordar.

El lenguaje Prolog tiene como herramientas a los llamados predicados, los cuales establecen relaciones entre objetos. Estos predicados se definen en términos de reglas lógicas y pueden representar una amplia gama de información, desde relaciones matemáticas hasta reglas de convivencia entre seres que habitan el planeta.

Para su ejecución es necesario usar consultas a la base de conocimiento, donde el intérprete de Prolog (que se tiende a representar como un búho, por el logo del lenguaje) busca todas las soluciones posibles, mediante el uso de unificación, backtracking y la aplicación de reglas.

La unificación se utiliza para igualar variables y términos para encontrar soluciones (javatpoint), y el backtracking permite explorar diferentes caminos para encontrar todas las soluciones posibles o para encontrar una solución única.

1.4. ANÁLISIS DEL PROBLEMA

Se reconocen cinco elementos principales, los cuales se organizan de la siguiente forma:

- Sistema: Es donde interactuarán los chatbots y los usuarios.
- Chatbot: Contiene los flujos de las distintas opciones a elegir.
- Flujo: Contiene las opciones elegidas por el usuario.
- Fecha: Entrega la fecha actual de cuando fue utilizado el programa.
- Usuario: Encargado de interactuar con los chatbots, tiene que registrarse antes de hacerlo e iniciar sesión (solo uno puede estarlo a la vez).

Los elementos sistema, chatbot y flujo tienen funcionalidades muy parecidas, como el agregar, no dejar que se agreguen duplicados entregando false, contar con ID’s para identificarlos, etc.

Por otro lado, el usuario debe poder registrarse y después lograr ingresar al sistema, pudiendo también cerrar sesión sin perder su registro, lo cuál estará guardado en el TDA ChatHistory.

Además dentro del sistema se debería lograr que los chatbots hablen entre sí con el predicado systemTalkRec, el cual será mostrado con el uso del predicado systemSynthesis y simulado con el uso del predicado systemSimulate.

Todo lo mencionado anteriormente deberá estar en sus respectivos TDA's, los cuales estarán en archivos distintos, junto a uno llamado "pruebas", el que contendrá todas las consultas que probarán al programa creado.

1.5. DISEÑO DE LA SOLUCIÓN

Para el problema expuesto anteriormente, se decidió hacer uso del concepto de TDA (tipo de dato abstracto) para poder hacer de una mejor forma la abstracción y construir una solución más apropiada al paradigma lógico. En el anexo 1 se encontrará un diagrama con los tres TDA's usados en la implementación, estos están presentados desde el elemento más pequeño del sistema (el cual sería el flujo) al sistema en sí que contiene todo lo que se usará.

Para llevar a cabo los TDA se usará el paradigma lógico en el lenguaje Prolog donde se construirán en el mismo orden descrito en el diagrama junto a cada componente necesario para su funcionalidad.

A diferencia del laboratorio 1, donde se trabajó con el paradigma funcional, a pesar de tener características iguales (como lo es el no agregar duplicados), no se pueden reutilizar los predicados entre los tres TDA, ya que contienen distintos recorridos, lo que fácilmente puede confundir al intérprete.

Al momento de hacer uso de los modificadores que agregan cierto elemento a algún constructor ya creado, se hará uso de recursión natural, la cual los agregará al final de la lista (en caso de no estar duplicados). Esto debido a que se debe recorrer la lista hasta cierto punto.

Cabe mencionar que además de los TDA, se tendrá un archivo main, donde se encontrarán los predicados principales y es donde se harán todas las consultas, ya que en él se usan todos los TDA's descritos anteriormente.

1.6. ASPECTOS DE IMPLEMENTACIÓN

El proyecto está separado en un archivo por cada TDA, junto a un archivo main.pl, el cual posee las funciones solicitadas en el enunciado. Estos archivos se conectan entre sí haciendo uso del module (exportador) y use_module (importador).

La versión de Prolog utilizada fue la 9.0.4, tanto de manera local como de manera remota, mediante el uso de la herramienta Swish-Prolog.

1.7. INSTRUCCIONES DE USO

En el archivo .zip se encontrarán todos los archivos .pl. Para poder hacer uso de ellos se debe abrir la consola de prolog y hacer un consult con el nombre completo del archivo main, después hacer uso de lo siguiente set_prolog_flag(answer_write_options,[max_depth(0)]), que prácticamente hace que la consola muestre las listas completas, cuando esto dé “true”, se ponen las consultas que se encontrarán en el archivo “pruebas”, se encontrarán comentadas las que no se alcanzaron a implementar y las que dan false, esto para evitar errores al momento de ejecutar el programa. Se dejará un ejemplo en el anexo 2.

1.8. RESULTADOS Y AUTOEVALUACIÓN

En general, el proyecto funciona a la perfección en lo que se logró avanzar, ya que los predicados que se hicieron funcionan con todas las pruebas realizadas.

Se logró implementar once de las catorce funciones exigidas debido al límite de tiempo y la falta de ingenio bajo presión para lograr hacer funcionar las faltantes.

Teniendo en cuenta lo anterior, la autoevaluación radica entre valores de 0 (no implementado) hasta 1 (implementado a la perfección) aumentando en una escala de 0.25. Esta se encontrará en el anexo 3.

1.9. CONCLUSIONES

En conclusión, los alcances que tiene el proyecto son parcialmente satisfactorios debido a que no se implementaron todos los predicados correspondientes, pero aún así los si logrados funcionan al 100% haciendo solo uso del paradigma lógico. La mayor limitación fue el cambio entre paradigma funcional a lógico, ya que al pensar en una solución, se hacía de una manera más efectiva en el paradigma funcional que en el lógico, por lo que costó entenderlo de otra forma y eso tomó su tiempo. También esto se debe a que el proyecto se comenzó a realizar en

primera instancia con Scheme, por lo tanto, el diseño de solución viene ya con un tipo de paradigma en bruto.

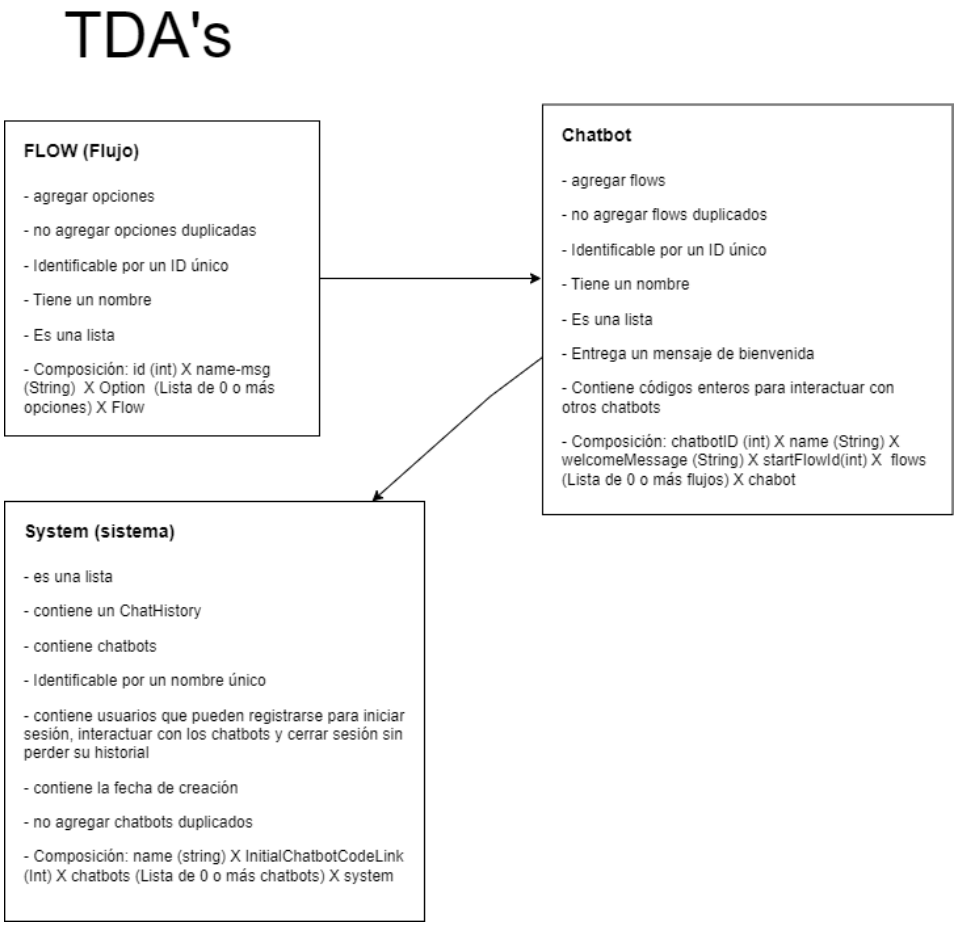
Por otro lado, es interesante ver como Prolog puede llegar a encontrar todas las soluciones posibles a distintas consultas debido a que está creado con el concepto de unificación y backtracking, distinto a como lo están otros lenguajes anteriormente vistos en la carrera, como C, Python, Scheme, etc.

1.10. BIBLIOGRAFÍA

javatpoint. (n.d.). Unification in Prolog. javatpoint. Retrieved November, 11, 2023, from https://www.javatpoint.com/unification-in-prolog

1.11. ANEXO

Anexo 1:



Anexo 2:

```
% c:/Users/Omen/OneDrive - usach.cl/Nivel 4/paradigmas/pruebas-lab-prolog/main_21060190_CastilloPerez.pl compiled 0.02 sec, 16 clauses
?-
|    set_prolog_flag(answer_write_options,[max_depth(0)]).
true.

?- option(1, "1 - viajar", 2, 4, ["viajar", "turistear", "conocer"], O1),
|    option(2, "2 - estudiar", 4, 3, ["aprender", "perfeccionarme"], O2).
O1 = [1,1 - viajar,2,4,[viajar,turistear,conocer]],
O2 = [2,2 - estudiar,4,3,[aprender,perfeccionarme]].

?-
```

Anexo 3:

Requerimiento no funcional / funcional	Puntaje
Autoevaluación	Obligatorio
Lenguaje	Obligatorio
Versión	Obligatorio
Standard	Obligatorio
Documentación	1
Dom->Rec	Obligatorio
Organización	1
Historial	1
Script de pruebas	Obligatorio
Prerrequisitos	Obligatorio
TDA	0.75
Option	1
Flow	1
flowAddOption	1
Chatbot	1
chatbotAddFlow	1
System	1
systemAddChatbot	1
systemAddUser	1
Systemlogin	1

Systemlogout	1
systemtalkrec	0
systemsynthesis	0
systemsimulate	0