UNIVERSIDAD SANTIAGO DE CHILE

Laboratorio de Paradigmas de Programación
Laboratorio Nº 2: ''Lenguaje Prolog: Sistema de Archivos"

Integrantes:
Claudio Hernández Hernández

Profesor:

Edmund Leiva Lobos

1. Introducción

El presente documento de carácter analítico tiene como objetivo que el estudiante logre realizar una explicación y resolución al problema presentado en el segundo laboratorio de Paradigmas de Programación. Para ello se debe realizar una breve descripción tanto del problema como del paradigma y conceptos que se usen en este proyecto, realizar un análisis de esto, donde se debe especificar los requisitos a cumplir, diseñar la solución explicando el enfoque que se decide tomar para resolver el problema, aspectos implementados para la resolución del mismo, ejemplos de instrucciones funcionales del programa, un resumen de los resultados que se obtienen, donde se explica cuales fueron exitosas y cuales no, y argumentar por qué fue así,, y por último, un cierre del trabajo hablando de las limitaciones, dificultades del paradigma, y una comparativa respecto al laboratorio trabajado anteriormente.

2. Descripción del Problema

En esta ocasión se debe realizar un Sistema de Archivos usando el Paradigma Lógico en el lenguaje de programación Prolog. Este sistema debe permitir una gran cantidad de funcionalidades, tales como registrar e iniciar sesión con algún usuario, la modificación de archivos y carpetas, restaurar elementos que hayan sido eliminados en la papelera de reciclaje, entre otras funciones que se mencionaran a lo largo del informe.
Por otra parte, hay requisitos que se deben cumplir a lo largo del trabajo, siendo algunos obligatorios para tener una evaluación mayor a la nota mínima, y otros opcionales. Un ejemplo de estos requerimientos obligatorios es el de realizar una autoevaluación de los requerimientos funcionales que son solicitados, usar una versión de SWI-prolog 8.4 o superior, entre otras.

3. Descripción Paradigma

Como se hizo mención en apartados anteriores, el paradigma que se debe implementar en este laboratorio es el de programación lógica. La programación lógica estudia el uso de la lógica tanto para plantear problemas, como para el control sobre las reglas que permiten concluir con una solución aritmética. Esta programación, al igual que la funcional, forman parte de la programación Declarativa, es decir, resolver problemas mediante sentencias. En la programación lógica se ejerce de manera descriptiva, estableciendo relaciones entre entidades para indicar el que se debe hacer.
En el Lenguaje de programación Prolog el código se compone de las siguientes partes:

1. Una Base de Conocimientos.
2. Términos que conforman el lenguaje.
3. Cláusulas que relacionan términos.
4. Hechos que permiten declarar valores verdaderos.
5. Reglas, las cuales son cláusulas que definen una verdad.

4. Análisis del Problema

Los requisitos solicitados en esta ocasión se pueden fragmentar en dos tipos:

1. Requerimientos No Funcionales.
2. Requerimientos Funcionales

En los Requerimientos No Funcionales hay 4 apartados que son obligatorios y 5 apartados opcionales. En los Requerimientos No Funcionales obligatorios tenemos:

1. Autoevaluación: como su nombre dice, se debe realizar una autoevaluación de los requerimientos funcionales solicitados.
2. Lenguaje: Se pide usar el lenguaje de programación Prolog en base a una programación mayormente declarativa.
3. Versión: Usar Swi-prolog versión 8.4 o superior.

4. Standard: No usar bibliotecas externas.

En cambio, en los Requerimientos No Funcionales opcionales se poseen:

1. Documentación: Realizar comentarios en el código que se realice.
2. Organización: Realizar los TDA en archivos independientes.
3. Historial: Commits en Github subidos de manera constante.
4. Script de pruebas: Archivo que documente el funcionamiento del programa.

Por otra parte, en los requerimientos funcionales se tienen:

1. TDAs: Realizar una debida implementación de los TDA pedidos, respetando la estructura dada.

Seguido de ello se muestran todos los TDAs que deben ser implementados, cada uno con su nombre de predicado, sus prerrequisitos y requisitos, su dominio y un ejemplo de cómo usarlo.

5. Diseño de la Solución

Antes de iniciar la resolución del problema, se decide hacer una revisión para estar más preparado a la hora de afrontar el laboratorio, por lo que se hizo un estudio del paradigma lógico con los recursos que contiene uvirtual, y una lectura tanto del tutorial de swi-prolog online y algunos enlaces externos que contiene.
Cuando ya se había realizado dicho repaso, se inicia la realización del proyecto. Lo primero que se decide hacer fue el escribir una base para todos los TDAs que se piden, escribiendo su dominio, recorrido, metas y una cláusula que solo hace referencia al predicado que se debe implementar.
A partir de eso el trabajo se fragmenta, ya que se avanza tanto en la programación como en el informe para aprovechar mejor el tiempo.
A medida que esos avances se realizan, también se prueban los TDAs ya terminados, para concluir si están bien terminados o les falta por finalizar.

6. Aspectos de Implementación

En este proyecto no se hizo uso de ninguna biblioteca externa en el código programado, debido a que esta explícitamente añadido en los requerimientos no funcionales que no se deben usar.
En cambio, para realizar el informe del proyecto se hizo uso del editor online Overleaf, usando un informe anterior como estructura para el nuevo.
Para los compiladores que se decidieron emplear, están SWI-Prolog (AMD646, Multi-threaded. versión 9.0.4), y también swish.swi-prolog.org.

7. Instrucciones de Uso

Al realizar las pruebas del programa, se decide el separar los TDAs por archivo independiente, realizar las pruebas correspondientes, y luego ir agregando los TDAs restantes para verificar si alguno de ellos hacia algún conflicto con otro.

1. Un ejemplo para el TDA-constructor es el realizar la consulta
   ?- system('Este es mi nuevo sistema', Sistema).
   Esta consulta recibe como respuesta
   Sistema = [system('Este es mi nuevo sistema', 1686616267.184499)].
   Se nos muestra el nombre del sistema que se ha creado, y la fecha en la que se hizo esa prueba.
2. En cambio, un ejemplo que se consultó y no dio entrega de lo esperado es en el TDA-login. La consulta es
   systemLogin([system('Este es mi nuevo sistema',1686616689.93616),['F', 'Disco Falco', 565420],['Nuevo Usuario']], 'Usuario 5', S3).
   En este caso el resultado a la consulta es

S3 = [system('Este es mi nuevo sistema',1686616689.93616),["F", "Disco Falco", 565420],["Nuevo Usuario"],["Login"]]
Se puede observar un error, ya que el usuario 'Usuario 5' no aparece registrado en el sistema, pero el sistema arroja de que el usuario se ha podido logear.

Hay muchos TDAs que no se mostró una implementación o ejemplo de ellos, por lo que se da una explicación en los siguientes dos apartados.

8. Resultado y Autoevaluación

Respecto a los resultados que se obtuvieron, se puede decir que los 6 primeros TDAs funcionan sin ningún problema, excepto por unos fallos con el TDA-login.
Sin embargo, los demás TDAs no se consigue ningún resultado, debido a que no están completados, por lo que solo dan un resultado true independiente de los argumentos que reciban. Esta problemática ha ocurrido por varios factores, pero el más importante es la mala organización del tiempo para realizar este proyecto.

9. Conclusiones del Trabajo

El lenguaje de programación Prolog al ser un programa muy simple, lo vuelve muy poco eficaz. Esto se hace más notar cuando al realizar los TDAs, no se reutilizan clausulas o reglas ingresadas en TDAs anteriores, provocando que trabajen cada uno de manera independiente.
Respecto al informe realizado en el proyecto anterior, se considera que este ha sido más elaborado y extenso con los temas que han sido tratados.