UNIVERSIDAD
DE SANTIAGO
DE CHILE

Laboratorio 2

Paradigmas de Programación

Sistema de Archivos

Nombre: Ka Hao Zeng Zhong

Rut: 20.642.718-3

Sección: A1

UNIVERSIDAD
DE SANTIAGO
DE CHILE

Índice de contenidos

1. Introducción

Para este trabajo se utilizará el paradigma lógico, el lenguaje de programación Prolog y el entorno de Swi-prolog el cual se ha visto en el curso.

1.1 Descripción breve del problema

Se debe desarrollar un programa basado en el lenguaje de programación Prolog para simular un sistema de archivos. Este programa debe permitir a múltiples usuarios realizar una serie de operaciones sobre archivos y carpetas, incluyendo la creación, modificación, compartición, copia, eliminación y encriptación, entre otros.

El sistema debe manejar varias unidades de almacenamiento, tanto físicas como lógicas, con un límite máximo de capacidad. Tanto las carpetas como los archivos tendrán atributos únicos como nombre, fecha de creación y modificación, permisos de usuario y tamaño. El programa también debe proporcionar una papelera de reciclaje para alojar temporalmente elementos eliminados, y permitir su restauración o eliminación definitiva.

1.2 Descripción breve del paradigma

El paradigma lógico pertenece al grupo de los paradigmas declarativos y se fundamenta en conceptos abstractos, especialmente en la lógica de primer orden. Un programa creado con un lenguaje fundamentado en este paradigma puede servir como una definición de un problema en lugar de proporcionar un conjunto de instrucciones para resolverlo.

En el paradigma lógico se define una base de datos o un conjunto de conocimientos que contiene hechos y reglas, y luego se pueden hacer consultas sobre esta base.

Prolog, el lenguaje de programación relacionado con el paradigma lógico se sustenta en tres mecanismos fundamentales: unificación, back tracking automático y estructuras de datos basadas en árboles. La ventaja principal de prolog y el paradigma declarativo en general es que permite centrarse más en la solución de un problema que en el proceso utilizado para llegar a la conclusión.

2. Desarrollo

2.1 Análisis del problema

Se debe contar con la capacidad de simular un sistema de archivos y poder manejar sus operaciones típicas, logrando a su vez la capacidad de manejar múltiples usuarios simultáneamente, así como la simulación de múltiples unidades de almacenamiento tanto físicas como lógicas.

Además, cada archivo y carpeta dentro del programa debe contar con una serie de atributos, así como sus propios predicados sobre sí que conllevan una serie de desafíos y complejidades y por último el sistema debe proporcionar una papelera de reciclaje para alojar temporalmente los elementos eliminados.

En resumen, el problema presenta una serie de desafíos en términos de simulación de un sistema de archivos, manejos de múltiples usuarios y unidades de almacenamiento, seguimiento de atributos de archivos y carpetas, soporte para una variedad de archivos y carpetas, y la implementación de una papelera de reciclaje. Cada uno de estos aspectos requerirán una atención cuidadosa durante el diseño e implementación del programa.

2.2 Diseño de la solución

Para resolver el problema, se debe definir cómo se representarán los usuarios, los archivos, las carpetas y las unidades de almacenamiento. Cada uno de estos elementos va a ser representado como un hecho en el código y cada uno tendrá sus propios atributos. Para cada operación que se puede realizar en archivos y carpetas se debe definir una regla correspondiente que tomen los argumentos necesarios y realicen la operación asignada.

También, para manejar múltiples usuarios, se puede manejar hechos y reglas donde cada usuario se identifique por un nombre único. De forma similar se trabajará con las unidades de almacenamiento donde se pueden representar como una lista de archivos y carpetas donde cada unidad tenga una capacidad máxima de almacenamiento e igualmente se implementará equivalentemente con los archivos eliminados por el usuario.

2.3 Aspectos de la implementación

Se utilizo Prolog versión 9.0.4 para desarrollar el programa, a través de un intérprete de este lenguaje, que puede ser tanto descargado como utilizado online. La estructura del código se dividió en dos archivos diferentes, uno correspondiente al archivo TDA y uno adicional que actúa como script de pruebas con ejemplos de requerimientos funcionales. Junto a los TDAs, se incluyeron predicados adicionales no obligatorios para mejorar la organización del código, todo debidamente comentado para evitar confusiones. Esta disposición del código se eligió tanto por cumplir con requisitos mandatorios, como facilitar la segmentación y orden de los requerimientos.

2.4 Instrucciones de uso

Para operar adecuadamente el programa, se necesita como requerimiento mínimo haber instalado o tener acceso a un interprete de Prolog, (versión 8.4 en adelante). Posteriormente, se debe indicar al interprete que consulte el archivo “pruebas.pl”. Si todo se hace correctamente, se puede iniciar el código y probar predicado por predicado con los parámetros especificados en la sección de requerimientos funcionales. Para facilitar la verificación de cada predicado, el archivo incluye una serie de ejemplos en forma de comentarios para cada uno de los predicados requeridos, incluyendo predicados anidados.

3. Resultados y autoevaluación

Al realizar cada predicado, se deben realzar varias pruebas para verificar su correcto funcionamiento. Este proceso se refleja en la siguiente tabla ya que, si una prueba produce un resultado inesperado o incorrecto, se descuenta la puntación. Los ejemplos utilizados están listados en el archivo de pruebas en forma de comentarios.

TDA system	Evaluación
Constructor	1
AddDrive	1
Register	1
Login	1
Logout	1
Switch-drive	0.75
Mkdir	0
Cd	0
Add-file	0
Del	0
Copy	0
Move	0
Ren	0
Dir	0
Format	0
Encrypt	0
Decrypt	0
Grep	0
View-trash	0
Restore	0

4. Conclusiones

En conclusión, tras haber trabajado en el programa, se ha demostrado ser una experiencia valiosa en términos de la lógica y el uso de Prolog, con el cual se interactuó de manera continua. También se enfrentaron distintos desafíos propios del paradigma, como la manera en que su aparente sencillez puede dar lugar a una complejidad cognitiva para el programador, lo que implica un obstáculo más mental que técnico.

Comparativamente, el paradigma lógico ha demostrado ser mas manejable en varios aspectos que el paradigma funcional, incluyendo ser mas intuitivo, ordenado, adaptable y generalmente más limpio. La utilización de variables simplifica notablemente el trabajo y ofrece un amplio margen de expansión al momento de formular predicados o funciones.