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

COMODÍN LABORATORIO II

Paradigma Lógico Aplicado En La Elaboración De Un Sistema Operativo de Archivos

Paradigmas de Programación - 13310
Profesor Roberto González Ibáñez, PhD.

Byron Caices Lima

24 de Julio, 2023

Tabla de Contenidos

1. INTRODUCCIÓN

En el presente informe se abordará el problema propuesto en el enunciado de laboratorio 2 acerca del paradigma lógico aplicado en el lenguaje de programación SWI-Prolog. Se revisará una descripción del problema y del paradigma empleado además del diseño de la solución y aspectos de la implementación.

2. DESCRIPCIÓN DEL PROBLEMA

Se presenta como desafío la creación de un sistema operativo de archivos (con orientación al usuario) en donde se tenga un sistema al cual se le pueden añadir unidades o drives y a estas unidades pueden añadirse carpetas y archivos implementando operaciones tales como añadir/formatear unidad, añadir/borrar/renombrar/encriptar carpeta u archivo. Se apunta a simular la consola del sistema introduciendo las operaciones típicas como cd, md, dir, etc. El principal desafío del paradigma radica en el cambio de mentalidad: de describir secuencias de instrucciones hacia la declaración de hechos y reglas que representan un problema.

3. DESCRIPCIÓN DEL PARADIGMA Y CONCEPTOS APLICADOS

En el campo de la programación, el paradigma lógico es notable por ser un miembro de los paradigmas declarativos. Este paradigma opera estableciendo una base de conocimientos que incluye hechos y reglas. Posteriormente, se hacen consultas a esta base de conocimientos.

Unificación: Se refiere al procedimiento de hallar un término o variable para hacer que una consulta sea válida o verdadera.

Consulta: Corresponde a una especie de "interrogante" que se plantea a la base de conocimientos. Se busca si la consulta puede unificarse con alguna parte de la base de conocimientos. Si no se encuentra ningún hecho correspondiente, se devuelve un resultado falso.

Hecho: Constituye una afirmación verdadera en la base de conocimientos, que se presenta con la primera letra en minúscula.

Predicado: Este es un concepto que expone una relación entre diferentes entidades y puede utilizarse para proclamar algo.

Átomo: En Prolog, los átomos son los elementos más básicos, son constantes que se pueden definir con “” o con su primera letra en minúscula.

Clausula: Una cláusula puede ser una regla o un hecho, y puede incluir varios hechos numerados. Todos estos hechos deben ser verdaderos para obtener un resultado verdadero.

4. ANÁLISIS DEL PROBLEMA

Dados los requisitos funcionales específicos que se deben cubrir lo primero que se debe realizar para abordar este problema es la identificación de los TDA necesarios. Notamos que se tienen 5 principales elementos para interactuar en un sistema de archivos: Sistema, Unidades, Carpetas, Archivos y Usuarios.

A su vez, este sistema está compuesto por atributos como un nombre, usuarios registrados, rutas accesibles (paths), unidades, contenido de las unidades y ese contenido posee carpetas y archivos, y las carpetas pueden poseer más carpetas y archivos "dentro". Al final, se asemeja a la estructura de árbol del Anexo 2. Sin embargo, visualizarlo como una estructura arbórea para la construcción del TDA puede ser un poco engorroso por lo que nos queda pensar en otra alternativa usando la estructura de datos más básica que nos ofrece Prolog: listas. Con esto se puede determinar que para manejar el sistema de archivos basta con trabajar con rutas; añadir una carpeta "Carpeta1" a una unidad "C:" no es más que agregar un path nuevo al sistema "C:/Carpeta1" y además agregar el TDA Carpeta al contenido del sistema para almacenar de alguna forma los TDAs que pueden ser trabajados en este. Sin embargo, no bastaría solo con agregar la ruta ya que nos faltaría saber, por ejemplo, la metadata de la carpeta o de un archivo como el usuario creador, fecha de creación, fecha de modificación y atributos de seguridad.

También quedaría definir qué es borrar un archivo en mi sistema, y se llegó a la conclusión de que borrar una carpeta o archivo es agregarlo a la papelera de reciclaje del sistema para que así ese ítem eliminado se vuelva inaccesible e inmutable pero que de todas maneras siga estando almacenado en la unidad en que se encontraba antes de ser borrado, tiene lógica ya que un sistema por sí solo no podría tener la capacidad de almacenar archivos, sino que estos son almacenados en una unidad.

Luego, considerando que el paradigma lógico unifica variables (no las modifica) tendremos Sistemas los cuales no pueden ser modificados directamente si no que se tendrá que reconstruir o unificar el sistema completo si es que queremos realizar una modificación. Dados estos fundamentos del análisis del problema queda pasar al diseño de la solución.

5. Diseño De La Solución

Como se mencionó en el punto anterior el TDA Sistema se compone de atributos tales como la hora actual, usuarios registrados, unidades, papelera y paths. La forma en que se decidió representar el sistema corresponde a una lista con listas, exceptuando aquellos valores que por su naturaleza no es necesario que sean una sublista del sistema como, por ejemplo, el nombre de este, la fecha y la ruta actual del sistema, los que para el caso de mi implementación serán un string. Luego, ya las listas que componen al sistema principalmente serán la sublista de paths que contiene todas las rutas accesibles y mutables del sistema (una lista de strings), la sublista que contiene la papelera que contendrá aquellos ítems (files/folders) que se vuelven inaccesibles ya que su fueron eliminados y finalmente la sublista de **contenido** se guardarán carpetas y archivos (a los que llamaremos items). Cada TDA Carpeta y Archivo tendrá el atributo location el cual se referirá a la ubicación del item.

En cuanto a los requerimientos funcionales, para realizar la implementación de systemDel se dividió el predicado principal en 4 diferentes cláusulas una con cada tarea y es que systemDel requiere eliminar: un archivo, una carpeta (y lo que contenga), todos los archivos de una carpeta con "*" o "*.*" y también eliminar archivos según su extensión con "*.ext". Este predicado requirió la implementación de 4 tipo de selectores de contenido del sistema que dado su CurrentPath, obtiene por ejemplo, los archivos que se encuentran en una ruta (getFilesFromFolder) o también dado el CurrentPath y un FolderName obtiene todo el contenido de dicho folder incluyendo subdirectorios y archivos contenidos (getFolderContent) y finalmente obtener archivos según su extensión desde un CurrentPath (getFilesByExtFromFolder). Con estos selectores el trabajo se "simplificó" bastante no solo para systemDel, sino que también para systemCopy y systemMove.

Para systemCopy se utilizaron los mismos selectores de contenidos permitiendo realizar todas las operaciones que el enunciado de laboratorio solicitaba y se tuvo que implementar un predicado para crear y setear las nuevas rutas que se están creando al agregar un nuevo ítem al contenido del sistema.

Con systemCopy funcionando nos damos cuenta de que para systemMove no es más que hacer la combinación de dos operaciones previamente implementadas como lo son Copy y Del, para mover una carpeta a una nueva ruta. Para mover un archivo a una nueva ruta se optó por la opción de buscar el archivo a mover en el sistema, cambiar el CurrentPath del sistema (creando así un nuevo sistema) y hacer un AddFile del archivo a mover pero en la nueva ruta, luego reseteamos el CurrentPath al que corresponde y se borra el archivo original con systemSupr (Es una modificación de systemDel que no envía ítems a la papelera).

6. Aspectos De La Implementación

Cada predicado de los requerimientos funcionales se encuentra documentada. La solución fue desarrollada en el lenguaje de programación Prolog con SWI-Prolog versión 9.0.4. No se utilizaron bibliotecas externas y se trabajó con el editor de código Visual Studio 1.80.1.

7. Instrucciones De Uso

Es esencial asegurarnos de que todos los TDA y el archivo de prueba se encuentren en la misma carpeta para asegurar su funcionamiento adecuado. Después de confirmar esto, abrimos el archivo de **pruebas** y se sugiere usar el comando "set_prolog_flag(answer_write_options,[max_depth(0)])." en la consola de consultas de Prolog, para una correcta visualización de los resultados.

Para utilizar el programa, debemos realizar las consultas en la consola proporcionada por el intérprete Swi-Prolog. Por ejemplo, si queremos generar un sistema, deberíamos hacer una consulta en la sección de consultas de esta forma: system("newSystem", S1). Esto consultará el predicado system, devolviendo True y unificando el sistema resultante

Posibles errores:

Un error que podría surgir al usar este programa podría deberse a introducir datos incorrectos en las consultas del programa. Sin embargo, el script de prueba funciona adecuadamente.

8. Resultados Y Autoevaluación

Después de realizar las pruebas con el script proporcionado en el laboratorio del paradigma lógico, este funcionó de manera satisfactoria en su totalidad. Además, se desarrollaron scripts de prueba para casos que deberían retornar false, y estos también operaron de manera correcta. Autoevaluación en Anexo 3

9. Conclusión

Se lograron los objetivos respecto a desarrollar un sistema operativo de archivos simplificado a través del paradigma lógico aplicado en una implementación en lenguaje Prolog. Fue un gran desafío iniciar este proyecto debido a la nueva forma de declarar todo como hechos en vez de instrucciones. Fue más abordable en algunos casos respecto al paradigma funcional, requería menos código que escribir. Se enfrentó una nueva forma de programar en donde no existían variables, ni ciclos for o while por ende se vio en la obligación de utilizar la recursividad, para conseguir las metas propuestas y cumplir con los requerimientos funcionales y no funcionales.

10. REFERENCIAS

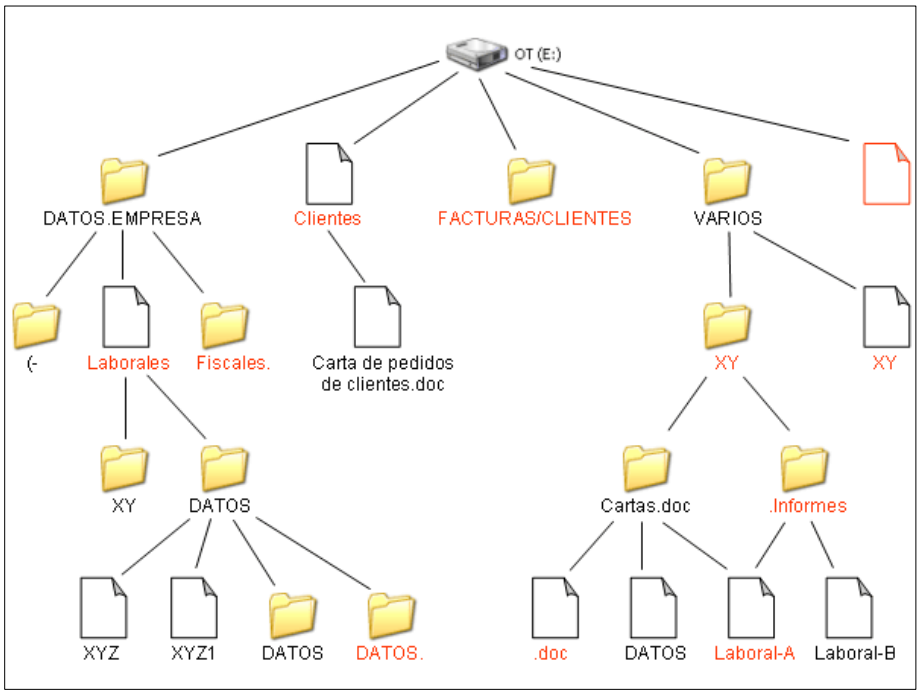
Navarro, S. Universidad San Jorge. (2021, 3 de junio). TIPS EN 2 MINUTOS - INFORMÁTICA: Sistema de archivos FAT. Recuperado de https://www.youtube.com/watch?v=bjUhkBBteRM

Soporte Software. (2014, 1 de abril). Sistemas de Archivos. Recuperado de https://www.youtube.com/watch?v=p2309jhCi9I

Aulapc (s.f). Archivos, Carpetas, Directorios, Arboles y Pathnames. Recuperado de http://www.aulapc.es/basico_archivos_carpetas.html

11. ANEXOS

Anexo 1: Estructura de árboles en un sistema de archivos



Anexo 2: Ejemplos de scripts

```
1 ?- set_prolog_flag(answer_write_options,[max_depth(0)]).
true.

2 ?- system("newSystem", S1), systemAddDrive(S1, "D", "Drive1", 1000000, S2), systemRegister(S2, "Usuario1", S3), systemLogin(S3, "Usuario1", S4), systemSwitchDrive(S4, "D", S5),systemMkdir(S5,"Carpeta1",S6),systemCd(S6,"/Carpeta1",S7),systemMkdir(S7,"Carpeta2"
,S8),systemRen(S8,"Carpeta2","NewCarpeta2",S9),writeln("% crea una carpeta1 y carpeta2 dentro de carpeta1, luego renombra carpeta2 por otro nombre, realizando sin probemas.").
% crea una carpeta1 y carpeta2 dentro de carpeta1, luego renombra carpeta2 por otro nombre, realizando sin probemas.
S1 = [newsystem,2023-07-24 19:46:09,[],,[],[],[],[],[]],
S2 = [newsystem,2023-07-24 19:46:09,[],,[],[[d,drive1,1000000]],[],[d:/],[]],
S3 = [newsystem,2023-07-24 19:46:09,[],,[usuario1],[[d,drive1,1000000]],[],[d:/],[]],
S4 = [newsystem,2023-07-24 19:46:09,[usuario1],,[usuario1],[[d,drive1,1000000]],[],[d:/],[]],
S5 = [newsystem,2023-07-24 19:46:09,[usuario1],d:/,[usuario1],[[d,drive1,1000000]],[],[d:/],[]],
S6 = [newsystem,2023-07-24 19:46:09,[usuario1],d:/,[usuario1],[[d,drive1,1000000]],[],[d:/,d:/carpeta1/],[[carpeta1,2023-07-24 19:46:09,2023-07-24 19:46:09,d:/,[usuario1]]]],
S7 = [newsystem,2023-07-24 19:46:09,[usuario1],d:/carpeta1/,[usuario1],[[d,drive1,1000000]],[],[d:/,d:/carpeta1/],[[carpeta1,2023-07-24 19:46:09,2023-07-24 19:46:09,d:/,[usuario1]]]],
S8 = [newsystem,2023-07-24 19:46:09,[usuario1],d:/carpeta1/,[usuario1],[[d,drive1,1000000]],[],[d:/,d:/carpeta1/,d:/carpeta1/carpeta2/],[[carpeta1,2023-07-24 19:46:09,2023-07-24 19:46:09,d:/,[usuario1]],[carpeta2,2023-07-24 19:46:09,2023-07-24 19:46:09,d:/carpe
ta1/,[usuario1]]]],
S9 = [newsystem,2023-07-24 19:46:09,[usuario1],d:/carpeta1/,[usuario1],[[d,drive1,1000000]],[],[d:/,d:/carpeta1/,d:/carpeta1/newcarpeta2/],[[carpeta1/,2023-07-24 19:46:09,2023-07-24 19:46:09,d:/,[usuario1]],[newcarpeta2/,2023-07-24 19:46:09,2023-07-24 19:46:09,
d:/carpeta1/,[usuario1]]]] 
```

```
4 ?- system("newSystem", S1), systemAddDrive(S1, "D", "Drive1", 1000000, S2), systemRegister(S2, "Usuario1", S3), systemLogin(S3, "Usuario1", S4), systemSwitchDrive(S4, "D", S5),systemMkdir(S5,"Carpeta1",S6),systemCd(S6,"/Carpeta1",S7),systemMkdir(S7,"Carpeta2"
,S8),systemMove(S8,"Carpeta2","D:/",S9),writeln("% creamos la carpeta Carpeta1 e ingresamos, creamos la carpeta Carpeta2 dentro de Carpeta1, movemos Carpeta2 a la raiz.").
% creamos la carpeta Carpeta1 e ingresamos, creamos la carpeta Carpeta2 dentro de Carpeta1, movemos Carpeta2 a la raiz.
S1 = [newsystem,2023-07-24 19:47:59,[],,[],[],[],[],[]],
S2 = [newsystem,2023-07-24 19:47:59,[],,[],[[d,drive1,1000000]],[],[d:/],[]],
S3 = [newsystem,2023-07-24 19:47:59,[],,[usuario1],[[d,drive1,1000000]],[],[d:/],[]],
S4 = [newsystem,2023-07-24 19:47:59,[usuario1],,[usuario1],[[d,drive1,1000000]],[],[d:/],[]],
S5 = [newsystem,2023-07-24 19:47:59,[usuario1],d:/,[usuario1],[[d,drive1,1000000]],[],[d:/],[]],
S6 = [newsystem,2023-07-24 19:47:59,[usuario1],d:/,[usuario1],[[d,drive1,1000000]],[],[d:/,d:/carpeta1/],[[carpeta1,2023-07-24 19:47:59,2023-07-24 19:47:59,d:/,[usuario1]]]],
S7 = [newsystem,2023-07-24 19:47:59,[usuario1],d:/carpeta1/,[usuario1],[[d,drive1,1000000]],[],[d:/,d:/carpeta1/],[[carpeta1,2023-07-24 19:47:59,2023-07-24 19:47:59,d:/,[usuario1]]]],
S8 = [newsystem,2023-07-24 19:47:59,[usuario1],d:/carpeta1/,[usuario1],[[d,drive1,1000000]],[],[d:/,d:/carpeta1/,d:/carpeta1/carpeta2/],[[carpeta1,2023-07-24 19:47:59,2023-07-24 19:47:59,d:/,[usuario1]],[carpeta2,2023-07-24 19:47:59,2023-07-24 19:47:59,d:/carpe
ta1/,[usuario1]]]],
S9 = [newsystem,2023-07-24 19:47:59,[usuario1],d:/carpeta1/,[usuario1],[[d,drive1,1000000]],[],[d:/,d:/carpeta1/,d:/carpeta2/],[[carpeta1,2023-07-24 19:47:59,2023-07-24 19:47:59,d:/,[usuario1]],[carpeta2,2023-07-24 19:47:59,2023-07-24 19:47:59,d:/,[usuario1]]]]
.
```

Anexo 3: Autoevaluación

RF			RNF	
Clases y Estructuras	1		Autoevaluacion	1
system	1		Lenguaje	1
add-drive	1		Version	1
register	1		Documentacion	1
login	1		Organizacion	1
logout	1		Diagrama de analisis	1
switchdrive	1		Diagrama de diseño	1
mkdir	1		git	1
cd	1			
addfile	1			
del	1			
copy	1			
move	1			
ren	1			
dir	0.75			
format	0			
encrypt	0			
decrypt	0			
grep	0			
viewTrash	0			
restore	0			