UNIVERSIDAD SANTIAGO DE CHILE

Laboratorio de Paradigmas de Programación
Laboratorio Nº 2: ''Simulación de un Sistema de Archivos en Prolog"

Integrantes:
Claudio Hernández Hernández

Profesor:

Edmund Leiva Lobos

Asignatura:

Paradigmas de Programación

1. Introducción

El presente reporte corresponde al segundo laboratorio del curso de Paradigmas de Programación. En esta ocasión, abordaremos el Paradigma Lógico, utilizando el lenguaje de programación Prolog a través del compilador SWI-Prolog.
En este informe, abordaremos un problema específico después de una breve descripción de este. A continuación, nos sumergiremos en la descripción del paradigma funcional y un conjunto de conceptos aplicados en este proyecto.
A medida que avanzamos, analizaremos en detalle el problema planteado y diseñaremos una solución acorde con el enfoque funcional. Posteriormente, compartiremos los aspectos más relevantes de la implementación del proyecto y proporcionaremos instrucciones y ejemplos para su uso.
Finalmente, mostraremos los resultados obtenidos y realizaremos una autoevaluación del trabajo realizado. Cerraremos este reporte con una conclusión que destaque los aprendizajes y las conclusiones obtenidas a través del desarrollo de este laboratorio.

2. Descripción del Problema

Se requiere crear una simulación de una plataforma que funcione como un sistema de archivos, como, por ejemplo, el Explorador de archivos de Windows. En esta plataforma, se permite la creación de unidades de almacenamiento, y seguido de ello el registro de usuarios, los cuales al identificarse e iniciar sesión correctamente tendrá la capacidad de crear tanto archivos como carpetas, editar el contenido realizado y compartirlo con otros usuarios que también están registrados en el sistema.

Para lograr esto, se deben tener en consideración algunos aspectos fundamentales:

1. Unidades lógicas: Corresponden a partes de almacenamiento que permiten acceder, gestionar y organizar de forma independiente. Cada unidad tiene una letra, un nombre y su capacidad. Estos valores permiten identificar a que unidad nos referimos y la cantidad de espacio que tengamos disponible para su uso.

2. Usuarios: Son a los perfiles que han sido ingresados en el sistema. Estos contienen un nombre de usuario que permite identificarlos. Al momento de iniciar sesión con uno de estos usuarios se le permite el uso de las operaciones disponibles en la plataforma.

3. Archivos: Ficheros creados por el usuario. Estos contienen el nombre del usuario que lo haya creado, fecha tanto de creación como de la última modificación realizada, sus atributos de seguridad y el contenido principal de dicho documento. Estos pueden ser manipulados a través de las operaciones disponibles.

4. Operaciones: Consideradas funciones que permiten realizar distintas acciones en el sistema, como por ejemplo el registro de usuarios, la creación de archivos, remover directorios, entre otros.

3. Descripción Paradigma

El paradigma lógico es un enfoque de programación que se basa en la lógica matemática y el razonamiento formal. En este paradigma, los programas se expresan como conjunto de reglas lógicas y hechos que se describen las relaciones y propiedades entre distintos objetos o conceptos.
El lenguaje de programación Prolog se utiliza en este trabajo, aunque no es un lenguaje puramente lógico, ya que también permite aspectos de programación declarativa y recursiva.
El paradigma lógico se basa en el lógica matemática, específicamente en el cálculo de predicados de primer orden y en el razonamiento lógico. Este se centra en la descripciones de relaciones y reglas lógicas para resolver problemas mediante la inferencia y el pensamiento lógico.
El paradigma lógico presenta elementos clave para la construcción de código utilizando este enfoque:

1. Declaratividad: Se enfoca en describir "qué"hacerse, en lugar del çómo"debe hacerse.

2. Predicados de primer orden: Se basa en el cálculo de predicados de primer orden, el cual permite cuantificar sobre variables y expresar relaciones entre objetos y conceptos mediante predicados y cláusulas lógicas.

3. Reglas y hechos: Las reglas establecen relaciones lógicas entre diferentes elementos, mientras que los hechos son declaraciones simples que se consideran verdaderas.

4. Recursividad: Técnica común en el paradigma lógico, permitiendo definir reglas y consultas que se aplican repetidamente para llegar a una solución.

4. Análisis del Problema

Como primer componente fundamental se tiene el sistema, el cual contiene a todos los demas elementos, como usuarios, documentos, directorios, entre otros. Esta plataforma es responsable de registrar y gestionar todos los cambios realizados de manera exitosa a través de operaciones específicas. Es el pilar fundamental que sostiene y organiza todos los datos y acciones que ocurren dentro del sistema.
Considerando esto, se requiere la implementación del sistema en Prolog, el cual puede ser representado de la siguiente manera:

Sistema: Se compone de un string y una fecha. A medida que los demás elementos y/o operaciones sean manipulados, este Sistema se ira actualizando con ello.

También se tienen más elementos importantes en el sistema:

- Unidades lógicas: Corresponden a las unidades de almacenamiento creadas antes del registro de usuarios. Estas listas almacenan una letra que representa su unidad, un nombre y su capacidad de almacenamiento (char x string x integer).
- Usuarios registrados: Usuarios que han realizado un registro en el sistema. Se compone de una lista con el nombre del usuario (string).
- Usuario logeado: Pertenece al usuario que ha iniciado sesión en el sistema. Solo contiene al usuario que se ha logeado (string).
- Directorio actual: Como su nombre dice, corresponde a una lista con la posición actual del usuario en el sistema. Cada acceso está representado por un string dentro de la lista (string x string x ...).
- Carpetas: Directorios que permiten almacenar archivos u otras carpetas dentro de ellas. Cada carpeta es representada con una lista. Estas contienen el nombre de la carpeta, dirección en la que se almacena, usuario que la crea, fecha de creación y de última modificación (string x string x string x list x list) x (string x string x string x list x list).
- Archivos: Ficheros almacenados dentro de la unidad lógica o de carpetas. Estos al igual que las carpetas también son representadas por listas. Estaslistas almacenan el nombre del archivo, el tipo de archivo, su contenido, directorio en el que ha sido creado, usuario que la crea, fecha de creación y de última modificación (string x string x string x lista x string x lista x lista) x (string x string x string x lista x string x lista x lista).
- Papelera: Como su nombre dice, representa una papelera de reciclaje. Aquí se almacenan tanto archivos como carpetas que hayan sido eliminadas, para así poder restaurarlas si se desea. Esta contiene sublistas que representan a los elementos eliminados (list x list x list x ...).

Por último, se tienen las funciones que se pueden realizar en la plataforma:

- system: Construye el sistema.
- systemAddDrive: Añade una unidad con letra única al sistema.
- systemRegister: Registra un usuario único en el sistema.
- systemLogin: Inicia sesión con un usuario existente en el sistema.
- systemLogout: Cierra sesión del usuario
- systemSwitchDrive: Fija la unidad en donde el usuario realizara acciones. Esta solo funciona si hay un usuario logeado.
- systemMkdir: Crea una carpeta con nombre especifico dentro de una unidad.
- systemCd: Permite cambiar la ruta en la que se realizaran operaciones.
- systemAddFile: Añade un archivo al sistema en la ruta actual.
- systemDel: Elimina un archivo. El elemento queda en la papelera.
- systemCopy: Copia un archivo en una ruta determinada.
- systemMove: Mueve un archivo a una ruta determinada.
- systemRen: Renombra un archivo o carpeta si el nuevo nombre es único en la ruta.

- systemDir: Lista el contenido de un directorio.
- systemFormat: Formatea una unidad, eliminando todo su contenido.
- systemEncrypt: Encripta un archivo o una carpeta con su contenido.
- systemDecrypt: Desencripta un archivo o una carpeta con su contenido.
- systemGrep: Buscar dentro del contenido de un archivo o de una ruta.
- systemViewTrash: Ver el contenido de la papelera.
- systemRestore: Restaurar contenido especifico de la papelera.

5. Diseño de la Solución

Al momento de diseñar la solución, además de utilizar datos nativos de Prolog, también se deben implementar datos específicos. Estos datos son creados haciendo uso de los llamados TDA, permitiendo la construcción de las operaciones de otros TDAs y para las operaciones mencionadas en el análisis.
El TDA más relevante en este proyecto es:

5.1. TDA SYSTEM

- Representado por una lista que contiene: un string y una fecha.
- Este TDA es construido con un nombre, y con ello se le asigna la fecha de registro.

Para la modificación y realización de los demás elementos que componen a la plataforma, se tienen las funciones disponibles en el sistema

5.2. OPERACIONES FUNCIONALES

- systemAddDrive: Función que realiza una verificación para validar que el segundo argumento sea un string de un carácter, o admitir como entrada un carácter, agregando la nueva unidad en caso de que no esté ya almacenada.
- systemRegister: Realiza una revisión para comprobar si el usuario ya está registrado. Si ya está dentro del sistema, devuelve el mismo sistema sin modificaciones. En caso contrario, agrega el nuevo usuario al sistema.
- systemLogin: Realiza una revisión para comprobar si el usuario ya está logeado. Si ya está logeado, devuelve el mismo sistema sin modificaciones. En caso contrario, inicia sesión con el usuario.
- systemLogout: Comprueba si hay un usuario ya logeado. En caso de que así sea, le quita la sesión al usuario. En cualquier otro caso, retorna false.
- systemSwitchDrive: Analiza si la unidad en la que se desea realizar acciones existe en el sistema, anotando en el sistema. En caso contrario, entrega false.
- systemMkdir: Función que comprueba si hay un usuario logeado en el sistema, y si la carpeta a agregar es única. Si es así, la agrega al sistema. Si no se cumplen las condiciones, retorna false.
- systemCd: Analiza que acción es la que se desea realizar, ya que tiene las opciones de regresar a una ruta anterior a la actual, regresar a la raíz de la unidad, ingresar a una carpeta o subcarpeta especifica, o quedarnos en la carpeta actual. Si no cumple ninguna de las anteriores, retorna false.
- systemAddFile: Realiza una comprobación si el nombre del archivo a ingresar ya está en el sistema. Si es el caso, se reemplaza ese fichero por el nuevo. En caso contrario, se añade al sistema.
- systemDel: Función que, dependiendo de la opción requerida, realiza una acción. Las opciones son la de borrar un archivo por su nombre, borrar todos los archivos por su extensión, borrar archivos que tengan una letra inicial y un formato específicos, borrar los archivos del directorio actual, borrar el contenido de una carpeta, incluyendo la carpeta. Si no se cumple ninguna de las anteriores, arroja false.

- systemCopy: Realiza una copia de un archivo, recibiendo la ruta de origen del archivo, y la ruta en la que será copiado. también tiene la función de copiar una carpeta o subcarpeta, copiar archivos con una extensión especifica o con una inicial especifica. En cualquier otro caso, retorna false.
- systemMove: Analiza si el archivo a mover ya existe en el destino, y si el archivo a mover existe. En caso de que en el destino ya exista un archivo con su nombre, lo debe sobrescribir. En caso de que no exista el archivo que se quiere mover, retorna false.
- systemRen: Realiza un cambio de nombre a un archivo especifico. Para ello debe verificar que el nombre al cual se quiere cambiar no existe en su mismo nivel. Si no cumple las condiciones, retorna false.
- systemDir: Lista el contenido de un directorio con distintas condiciones, ya que puede listar el contenido del directorio actual, el actual y de sus subdirectorios, listarlas en orden alfabético, entre otros. Si no cumple ninguna de las posibilidades, retorna false.
- systemFormat: Para poder formatear una unidad, primero necesito verificar que esa unidad existe en el sistema. En el caso que, si exista, borra todo su contenido, permitiendo además cambiarle su nombre.
- systemEncrypt:
- systemDecrypt:
- systemGrep:
- systemViewTrash:
- systemRestore:

6. Aspectos de Implementación

Este laboratorio utiliza el compilador SWI-Prolog, más específicamente las versiones 8.4 o superiores a esta.
La implementación es principalmente con el uso de listas. Sin embargo, se considera el no uso de funciones tales como integer o is_list en la elaboración de los TDAs, ya que estas pertenecen a átomos representativos, por lo que se pide encapsular estas funciones dentro de los TDAs creados. La estructura del Código es en archivos independientes, donde uno de ellos se considera el archivo principal que implementa una importación de las operaciones y TDAs del código, permitiendo la ejecución de todas las funciones. Este archivo también el script de pruebas en sus comentarios.
En total se tienen 21 archivos, los cuales siguen el siguiente nombre:

- "NombreTDA-rut-ApellidoPaterno-ApellidoMaterno.pl"

Exclusivamente, hay uno de los archivos que no contiene el mismo orden que los demás, ya que al ser el archivo principal se tiene con el nombre "main.pl".

7. Instrucciones de Uso

Primero se debe comprobar que se tienen todos los archivos en la misma carpeta. Seguido de ello, se debe ejecutar el programa SWI-Prolog, con el cual se puede abrir el archivo "main.pl"para realizar las consultas solicitadas.

7.1. Resultados Esperados

Se busca desarrollar una simulación de un sistema de archivos sin problemas ni ambigüedades. La implementación debe estar libre de errores y utilizar de manera adecuada las estructuras de listas y la recursión, que son elementos fundamentales para el desarrollo del proyecto.
Cada funcion debe realizar su tarea correctamente y los Tipos de Datos Abstractos (TDAs) deben tener representaciones adecuadas y bien definidas. En otras palabras, se espera un programa robusto, eficiente y bien organizado que permita un sistema de archivos fluido y sin errores.

8. Resultado y Autoevaluación

Los resultados obtenidos fueron los esperados, ya que se crearon las funciones obligatorias y una gran parte de las operaciones funcionales.
Los errores que se pueden esperar son debido a las funciones que no se lograron realizar completamente, debido al ajustado tiempo restante para la entrega del comodín.

Se realizaron múltiples pruebas con distinto ejemplos para comprobar que las funciones no arrojen algún error en la ejecución del Código. Se lograron completar exitosamente 13 de las 21 funciones.

8.1. Autoevaluación

Es una lista que se realiza de la siguiente manera:

- 0: No realizado
- 0.25: funciona 25 % de las veces
- 0.5: funciona 50 % de las veces
- 0.75: funciona 75 % de las veces
- 1: funciona 100 % de las veces

8.2. Requerimiento No Funcionales

1. Autoevaluacion:1
2. Lenguaje:1
3. Version:1
4. Standard:1
5. Documentacion:0.75
6. Organizacion:1
7. Historial:1
8. Script de pruebas:0.5
9. Prerrequisitos:1

8.3. Requerimiento Funcionales

1. TDAs: :1
2. system:1
3. systemAddDrive:1
4. systemRegister:1
5. systemLogin:1
6. systemLogout:1
7. systemSwitchDrive:1
8. systemMkdir:1
9. systemCd:0.5
10. systemAddFile:1

11. systemDel:1
12. systemCopy:0.5
13. systemMove:0.25
14. systemRen:1
15. systemDir:0.25
16. systemFormat:0.25
17. systemEncrypt:0.25
18. systemDecrypt:0.25
19. systemGrep:0.25
20. systemViewTrash:0.25
21. systemRestore:0.25

9. Conclusiones del Trabajo

Después de finalizar y concluir el proyecto, se puede afirmar que se lograron cumplir los objetivos principales de manera satisfactoria. Durante el proceso, se adquirió una mejor compresión del uso de Prolog y SWI-Prolog, lo que permitió realizar un gran avance del laboratorio en comparación de la primera entrega. Además, se obtuvo un aprendizaje significativo sobre un nuevo paradigma de programación y se pudo aplicar de manera efectiva, aplicando los conceptos fundamentales aprendidos en clase, como el uso de TDAs, recursividad, unificación, currificacion, reglas y hechos, entre otros.
La realización del proyecto presento un desafío significativo debido a la naturaleza de las ideas implementadas. En ocasiones, algunas ideas no funcionaban como se esperaba, lo que requería la revisión y ajuste de enfoques. Lo normal en caso de que ocurriera uno de estos problemas, se optó por realizar cambios cuando era necesario para un correcto funcionamiento. Este enfoque de iterar y adaptarse fue esencial para lograr un proyecto exitoso y funcional.

Nota final esperada: 5.4