

**Universidad de Santiago de Chile**

**Facultad de Ingeniería**

**Departamento de Ingeniería en informática**

Laboratorio 2 – Paradigmas de Programación

Paradigma Lógico - Prolog

Profesor:

Gonzalo Martínez Ramírez

Integrante:

Felipe Solís Nahuel

Índice

1. Introducción. 3
2. Descripción del paradigmas y conceptos aplicados. 3
3. Análisis del problema y requisitos que se deben cubrir. 4
4. Diseño de la solución. 6
5. Aspectos de implementación. 7
6. Instrucciones de uso. 7
7. Conclusión. 8
8. Referencias. 9

INTRODUCCIÓN

Para este proyecto semestral se trabajarán distintos paradigmas de programación, los cuales serán abordados en cuatro lenguajes de programación por lo que serán cuatro entregas de laboratorio. Para el desarrollo satisfactorio del curso se requerirá desarrollar un sistema para la administración y editor de documentos colaborativos como lo es GoogleDocs. Estos documentos colaborativos permitirán a uno o más usuarios compartir documentos y hacer ediciones sobre este. En GoogleDocs se permite crear y administrar documentos de distinto tipo. Para este laboratorio nos encontraremos solo con el editor de texto y con los permisos de usuario para crear o realizar actividades en los documentos de la plataforma a los cuales ha creado o se le han compartido, al compartir los documentos se le pueden indicar los privilegios de usuario sobre el documento, específicamente pueden tener los permisos para leer, comentar, editar y compartir. Las ediciones sobre un documento, dará a lugar una nueva versión del documento, por ello todas las versiones de un documento conforman lo que se denomina “historial de versiones”.

DESCRIPCIÓN DEL PARADIGMAS Y CONCEPTOS APLICADOS

En este laboratorio se trabajará a través del paradigma lógico (tipo de lenguaje declarativo), en que se basa en razonar para resolver problemas como lo son las matemáticas,

en el conocimiento básico de éstas se representan en la lógica en forma de axiomas a los que se les añaden reglas para deducir verdades. La programación lógica históricamente viene ligada del lenguaje llamado Prolog (PROgrammation LOGique) el cuál será el lenguaje que se deberá ocupar para el siguiente laboratorio. Los conocimientos sobre el problema se expresan en forma de predicados (axiomas) entablando relaciones entre los símbolos que representan los datos del dominio del problema.

El laboratorio se trabajará con los conceptos de TDA (Tipos de Datos Abstractos) junto con los conceptos de paradigma lógico con sus mecanismos básicos: Unificación, Backtracking automático y estructuras de datos basadas en árboles.

Los TDA’s, en su implementación, corresponden a un “qué” se debe implementar y no el “cómo” se debe implementar, por ello se trabajan por medios de:

Constructores: Son la representación conceptual a su implementación con recursos provistos de un lenguaje de programación (en este caso Prolog). Aquí se permiten generar elementos particulares que se ajusten a una representación.

Pertenencias: Son operaciones que permiten verificar si un elemento pertenece o no a un determinado conjunto/tipo de dato, el retorno de estas debe ser de tipo booleano.

Selectores: Estos permiten obtener partes individuales de un tipo de dato compuesto, para ello deben conocer la representación interna, de las que tienen dos formas de obtenerlas:

* Selectores naturales: obtienen los elementos directos.
* Selectores derivados: obtienen los elementos sustentados por los selectores naturales.

Modificadores: Estos alteran los datos de un determinado elemento a lo que puede llegar a la mutación de los elementos o a la creación de estos. Ellos, al igual que los selectores, pueden conocer la representación interna de las representaciones.

Otras operaciones: operaciones y acciones que pueden realizar un TDA, estos pueden ser apoyadas por los niveles anteriores o directamente por la representación interna del TDA.

ANÁLISIS DEL PROBLEMA Y REQUISITOS QUE SE DEBEN CUMPLIR

Para realizar este laboratorio se deberá seguir los principios de este paradigma respetando las siguientes solicitudes requeridas:

* Requerimientos no funcionales:

- Autoevaluación: Incluir autoevaluación de cada uno de los requerimientos funcionales solicitados.

- Lenguaje: La implementación debe ser en el lenguaje de programación Scheme.

- Versión: Usar Swi-prolog versión 8.x.x .

- Documentación: Todos los predicados deben estar debidamente comentados. Indicando descripción de la relación, términos de entrada y de salida.

- Historial: Historial de trabajo en Github tomando en consideración la evolución en el desarrollo de su proyecto en distintas etapas.

- Ejemplos: Al final de su código incluir al menos 3 ejemplos de uso para cada uno de los predicados correspondientes a requerimientos funcionales obligatorios y los extras.

- Prerrequisitos: Para cada predicado se establecen prerrequisitos. Estos deben ser cumplidos para que se proceda con la evaluación de la funcionalidad implementada.

* Requerimientos funcionales:

- TDAs: Implementar abstracciones apropiadas para el problema. Analizar el problema y determinar el o los TDAs y representaciones apropiadas para la implementación. Luego, planificar bien el enfoque de solución de manera que los TDAs y representaciones escogidas sean aplicables a ambos tipos de funciones.

- Register: Predicado que permite consultar el valor que toma un TDA *ParadigmaDocs* al ocurrir el registro de un nuevo usuario a la plataforma de documentos. Para esto se solicita la plataforma de documentos inicial, nombre del usuario, contraseña y el *ParadigmaDocs* resultante luego del registro. El retorno del predicado es *true* en caso de que se pudo satisfacer el registro del usuario.

- Login: Predicado que permite autenticar a un usuario registrado. Si la autenticación es válida, el retorno es true. La actualización del ParadigmaDocs incorpora al usuario autenticado en sesión activa (fundamental para que los predicados siguientes puedan funcionar).

- Create: Predicado que permite a un usuario con sesión iniciada en la plataforma crear un nuevo documento. Cada documento registra el autor de la misma, fecha de publicación, el nombre del documento y el contenido del documento. El retorno es *true* si se puede satisfacer en “Sn2” el TDA *ParadigmaDocs* con el nuevo documento incluido y sin la sesión activa del usuario que creó el documento.

- Share: Predicado que permite a un usuario con sesión iniciada en la plataforma compartir a otros usuarios un documento especificando el tipo de acceso a éste. El retorno es *true* si se puede satisfacer en “Sn2” el TDA *ParadigmaDocs* con el archivo a compartir especificando los usuarios, los permisos con que está compartido aquel archivo y sin la sesión activa del usuario.

- Add: Predicado que permite a un usuario con sesión iniciada añadir texto al final de la versión actual/activa del documento. En el TDA resultante debe estar **una nueva versión activa del documento con el texto que se agregó al final.**

- RestoreVersion: Predicado que permite a un usuario con sesión iniciada restaurar una versión anterior de un documento. En el TDA *ParadigmaDocs* resultante la versión activa pasa a ser una versión más dentro del historial y la versión restaurada pasa a ser la versión activa del documento.

- ParadigmadocsToString: Predicado que permite obtener una representación de un TDA *ParadigmaDocs* como un string posible de visualizar de forma comprensible al usuario. Debe hacer uso del *char* ‘\n’ para los saltos de línea. **No utilice los predicados *write* y *display* dentro de este requerimiento**, ya que debe quedar en el último argumento un string el cual pueda luego ser pasado como argumento a los predicados “write” o “display” para poder visualizarlo de forma comprensible al usuario.

DISEÑO DE LA SOLUCIÓN

Para el diseño del código a formular tendremos en cuenta lo anterior aprendido en el laboratorio 1 realizado con Scheme, la idea es similar comenzar identificando los TDA’s de la situación formalizándolos como predicados y sus debidos selectores y modificadores que serán fundamentales para la creación de los requerimientos funcionales. Como resultados de esta sección obtendremos los TDA’s de usuario, fecha, documento y paradigmadocs, con contenido parecido a los TDA’s formulados en Scheme con leve diferencia.

- TDA fecha, generará una lista que contenga día, mes y año del elemento a representar.

- TDA usuario contendrá un nombre, una contraseña y un id con el se podrá identificar más fácilmente al usuario y trabajar con el no sea tan difícil.

- TDA documento contendrá su propio id, un nombre, un contenido, una fecha creada por el TDA fecha, una lista con los permisos de cada usuario y por último una lista donde se guardará las versiones anteriores del documento.

- TDA paradigmadoc, es aquí en donde se contendrá los elementos creados gracias a las reglas (requerimientos funcionales) con tal de operar y almacenar más datos, por ende éste deberá contener su nombre, una fecha creada, una lista de usuarios registrado, una lista de documentos creados, un historial de versiones anteriores de la plataforma, un id que representará la cantidad de usuarios que hay (el cual también servirá como indicador para los id de cada usuario) y por último una lista de la sesión activa que servirá cuando se realice la operación login guardar el usuario logueado para poder indicar que alguien está en la plataforma y puede operar en ella.

Con estos TDA’s será suficiente para operar, en cambio para los requerimientos se parte de manera ordenada.

ParadigmaDocsRegister, como se explicó en la sección anterior, será una función que nos permita verificar si el usuario se registró satisfactoriamente, para ello se debe verificar que el usuario no esté registrado, de lo contrario el código arrojará false. Luego de que se verificase, se debe añadir el nuevo usuario con su nombre, contraseña y fecha, el id se dará de manera automática por medio del indicador que se encuentra en la plataforma, luego el usuario se agrega a la lista de usuarios y es reemplazada por la lista anterior a través de una regla, al igual que el id se le debe sumar un 1 para luego ser reemplazada, al igual que la lista de usuarios, en el paradigmadoc y así entregar una nueva plataforma.

ParadigmaDocsLogin, en este se debe verificar que el usuario se encuentra dentro de la lista de usuarios, para verificarlo el usuario entrega el nombre de usuario y la contraseña con el cual por medio de una regla podrá buscar el usuario que contenga los datos anteriores, una ves encontrado se crea un nuevo paradigmadocs en el que la sesión activa será reemplazada por el usuario indicado.

ParadigmaDocsCreate, en esta regla se entrega el paradigmadocs actual, una fecha, el nombre del documento, y el contenido. Del paradigmadocs actual se obtiene el id del usuario con la sesión actual para crear un permiso que será guardado dentro de la lista de permisos del documento, luego se revisa el documento creado anteriormente para obtener su id y con su sucesor añadirlo al documento que se está creando al igual con el resto de los datos que han sido entregados por parte del usuario, una vez creado, se añade a la lista de documentos de la plataforma.

ASPECTOS DE IMPLEMENTACIÓN

Para poder trabajar con este código deberá tener acceso a la consola de Prolog con versión superior a la 8.0.0, solo se contará con las librerías básicas de este, una manera más factible si cuenta con internet, dirigirse a la página swish prolog en el que fue trabajado este código, las razones por las que fue tr.

INSTRUCCIONES DE USO

Para poder trabajar con este programa, principalmente se deberá crear un paradigmadoc (pude ser más de uno si lo desea) con cada uno de estos ingresando, los cuales deberán entregarles un **nombre de la plataforma**, un **TDA fecha válida** las cuales puede ser distinta o iguales para todas, y por último un **nombre para el hecho** (los cuales deben empezar con una mayuscula) resultante el cual será nuestra(s) plataforma(s). Ejemplo de como se vería según lo anterior:

fecha(21,09,2021, Fecha1).

paradigmadoc("paradigmadoc1", Fecha1, PD1).

Para registrar un usuario deberá ingresar la regla paradigmaDocsRegister y entre paréntesis ingresar el paradigmadoc anterior, un **TDA fecha**, un **nombre de usuario**, una **contraseña**, y por último ingresar un **nombre que identifique al elemento** resultante que será un nuevo paradigmadoc con un nuevo usuario en la lista de usuarios registrados. Ejemplo:

paradigmaDocsRegister(PD1, Fecha1, “MarcoAntonioSolis”, “chayane123”, PD2).

Si desea verificar, debe ingresar la lista resultante y reemplazarla por PD2.

Para loguear un usuario se deberá llamar la regla paradigmaDocsLogin, en el que dentro de un paréntesis se ingresará: **paradigmadoc anterior**, **TDA fecha**, **nombre de usuario**, **contraseña del usuario** (nombre y contraseña deben estar dentro de la lista de usuarios, sino la operación retornará un false) y un **nombre para la lista resultante**. Ejemplo:

paradigmaDocsLogin(PD3, "MarcoAntonioSolis", "chayane123", PD4).

Para verificar que la lista resultante es correcta debe ser ingresada reemplazando el nombre que se le dio a la lista, en el caso anterior PD3.

Para crear un documento se llama la regla paradigmaDocsCreate, por consiguiente, entre paréntesis ingresar: **paradigmadoc anterior**, **TDA fecha**, **nombre del documento**, **contenido del documento** y por último un **nombre para la lista resultante**, paradigmadoc que contendrá el documento con sus datos ingresados en la lista de documentos. Ejemplo:

paradigmaDocsCreate(PD5, Fecha2, "Primer Documento", "hola123", PD6).

RESULTADOS Y AUTOEVALUACIÓN

Principalmente a destacar es que el código no cumple con todos los requerimientos solicitados, pese a ello igual es posible trabajar con el aplicando funciones como register, login y create las cuales son las funciones básicas de la plataforma, al igual que los TDA’s que fueron bien aplicados exceptuando que ciertos elementos que contienen estos no fueron utilizados, como por ejemplo la lista de historial de los documentos y paradigmas, pese a todo es posible trabajar correctamente los requerimientos que se lograron hacer sin ningún problema cumpliendo los prerrequisitos y requisitos de implementación de cada predicado.

CONCLUSIÓN

El objetivo de este laboratorio fue recrear una plataforma parecido a lo que es el programa googledocs, en el que se puede trabajar con documentos y compartirlos para trabajar mutuamente, específicamente, lograr que un documento se vaya actualizando para los demás usuarios (a quienes se les fue compartido) para poder trabajarlo en conjunto y por cada cambio se actualice la versión para los demás y puedan visualizar sus cambios u volverlos a trabajar. Las limitaciones que se tuvo para este laboratorio fue principalmente la falta de tiempo, es por ello por lo que no se logró entregar más requerimientos funcionales, pero aún así en lo que se logró en requerimientos funcionales se logró correctamente junto con todos sus requisitos de implementación.

A diferencia del laboratorio anterior, el nivel de complejidad de programar el código fue bastante similares, por el cambio de paradigmas un poco drástico dificultó un poco el entendimiento de este ya que se necesita tener una mentalidad más amplia para empatizar con el nuevo paradigma, aún así se logró entender perfectamente, además de la atención que se le debe dar al código en Prolog es un tanto más fácil que en Scheme por la cantidad de paréntesis o por pequeños problemas que pueden estar de por medio, pero Prolog tampoco se escapa de su dificultad con su nivel de programación bastante analítica en sus recursiones que se les debe tener mucha atención para que se ejecute correctamente a lo que es pensado.

REFERENCIAS

Serra, S. (2009). *Paradigmas de Programación - Programación Lógica*. Universidad Tecnológica Nacional Facultad Regional Córdoba.