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

INFORME DE LABORATORIO 2: SIMULACIÓN DE UNA PLATAFORMA ESTILO "GOOGLE DOCS" EN PROLOG

Nombre: John Serrano C.
Profesor: Roberto Gonzales I.
Asignatura: Paradigmas de Programación

03 de enero 2022

Tabla de Contenidos

1. INTRODUCCIÓN

El presente informe corresponde al laboratorio número 2 del curso de Paradigmas de Programación. Este segundo laboratorio corresponde al Paradigma Lógico, donde se utiliza el lenguaje de programación Prolog a través del IDE SWI-Prolog. El informe sigue la siguiente estructura: Primero, se tiene una descripción breve del problema, luego una descripción del paradigma, los objetivos del proyecto, análisis del problema, diseño de la solución del problema, aspectos de implementación, instrucciones y ejemplos de uso, resultados y autoevaluación y finalmente una conclusión al proyecto realizado.

1.1 DESCRIPCIÓN DEL PROBLEMA

Se pide desarrollar una simulación de una plataforma de edición de texto colaborativo, como, por ejemplo, Google Docs. Esta plataforma permite registrar usuarios y estos al loguearse correctamente tienen la capacidad para crear un documento junto con su texto, editar el mismo y compartir el documento con otros usuarios registrados en la plataforma, entre otras cosas. Para implementar esto, se deben tener algunos elementos en cuenta:

Usuarios: Corresponden a los usuarios registrados en la plataforma. Cada usuario tiene un username y un password, que funcionan como las credenciales de este, junto con su fecha de registro en la plataforma. Los usuarios registrados y logueados correctamente pueden hacer todas las operaciones disponibles en la plataforma.

Documentos: Estos son creados por los usuarios y registran a un autor, una fecha de creación y un nombre, junto con el contenido principal del documento. Estos pueden ser compartidos con otros usuarios y todos los cambios del contenido del documento quedan registrados en el historial de versiones. Tienen un ID único.

Operaciones: Son aquellas que registran y loguean a un usuario y crean, comparten, editan y modifican a un documento en específico, entre otras cosas.

Todo debe ser implementado utilizando el Paradigma Lógico, a través de la programación en Prolog.

1.2 DESCRIPCIÓN DEL PARADIGMA

El Paradigma Lógico se basa en la definición de reglas lógicas y es un paradigma declarativo. Contiene una base de conocimientos, la cual es un archivo que contiene los predicados del programa y esto es “todo el mundo” que conoce. Los predicados son cosas que se quieren decir y estos están definidos por clausulas (hechos y reglas).

Los hechos y/o reglas a su vez son definidos por “Individuos” o “Átomos” que son una representación de lo que se quieres definir como base (ej. name(victor)). Se debe tener en cuenta que tanto hechos como reglas comienzan con minúsculas.

En el Paradigma Lógico se tienen algunos elementos bastante importantes a la hora de construir un código utilizando este paradigma, como, por ejemplo:

Átomo: son aquellas cosas sobre las que basa el conocimiento que queremos expresar (Se escriben en minúsculas).

Predicado: Los predicados son las cosas que queremos decir. Los resultados o variables van en Mayúsculas.

Consultas: Son aquellas preguntas que se hacen mediante consola y donde Prolog busca en la base de conocimientos para entregar True o False. También puede entregar un elemento que satisfaga una consulta para que sea verdadera, en el caso de que se haya introducido una variable.

Cláusulas de Horn o hechos: Cada una de las sentencias se "iguala a" unidades de información de una base de conocimiento. Los hechos y reglas deben terminar con un punto.

Además, se debe evitar el Problema de mundo cerrado que se puede generar al negar un predicado o un hecho y a su vez que el resultado que se espere no se haya definido por lo que no es nada de la base de conocimiento, que en teoría podría ser cualquier cosa. Cabe destacar que cuando Prolog da falso a una respuesta, no significa que sea un falso absoluto, si no que cuando pasa esto, significa que se dio el caso en que Prolog no fue capaz de encontrar un hecho y/o resultado que satisfaga la pregunta y por lo tanto, retorna falso.

Para la base de conocimientos, no es necesaria que esta contenga todo un conjunto de información, si no que contenga solo la información necesaria para satisfacer una respuesta a un problema. Muchos problemas computacionales pueden ser expresados en términos de Cláusulas de Horn y resueltos a través del Paradigma Lógico.

1.3 OBJETIVOS

Como objetivos del proyecto se tiene aprender sobre el Paradigma y la programación lógica, para así obtener la habilidad de programar de otra forma distinta a la que se tiene costumbre actualmente. Otro objetivo es programar correctamente en Prolog y aprender a utilizar las herramientas de este lenguaje para completar el proyecto de laboratorio y así poder tener una base para los futuros laboratorios y otros proyectos que en un futuro se desarrollen con la Programación Lógica.

2. DESARROLLO

2.1 ANÁLISIS DEL PROBLEMA

Cabe destacar de que en el proyecto no se trabaja con Bases de Datos o similares, ya que son conceptos que aún no se han introducido y, por lo tanto, solo se requiere realizar una simulación de la plataforma a través de consultas a la base de conocimientos de Prolog. Entendiendo esto, se tienen algunos elementos fundamentales en el problema:

Primero, la plataforma en sí, la cual es la base de todo y donde se guardan todos los demás elementos importantes, como son los usuarios, documentos, permisos, versiones de los documentos, entre otras cosas. La plataforma registra todos los cambios que se realizan exitosamente a través de una operación. Teniendo en cuenta esto, se requiere representar a la plataforma en Prolog, donde se puede representar de la siguiente manera:

Plataforma: Lista que contiene a un string, una fecha y tres listas donde se irán guardando los usuarios registrados, el usuario activo y los documentos respectivamente. (string X fecha X list X list X list).

A diferencia del primer laboratorio, para este no es necesario trabajar con funciones de encriptación para el texto de las contraseñas y documentos.

Luego, se tienen otros elementos importantes que se guardan en la plataforma:

- **Usuarios:** Corresponden a aquellos que interactúan con la plataforma, registrándose en esta para luego poder realizar las operaciones disponibles.
- **Documentos:** Son creados por los usuarios y tienen información escrita registrada en el documento. Los documentos se guardan en la plataforma y contienen información de identificación al igual que un ID único para poder diferenciar un documento de otro.
- **Permisos:** Corresponde a un conjunto de asociaciones, donde a un usuario se le da un grado de acceso a un documento. Existen cuatro tipos de permisos: Lectura, Escritura, Comentarios y Compartir. Se representan con “R”, “W”, “C” y “S” respectivamente.
- **Historial de versiones:** Es donde queda registro de las versiones de un documento. Una versión nueva se produce cuando se realiza algún cambio en un documento, ya sea editar texto, eliminar texto, cambiar texto, etc. Cada versión tiene identificación. El historial de versiones de un documento se guarda en el propio documento.

Para poder implementar a los elementos mencionados anteriormente en Prolog, se pueden representar de la siguiente manera:

- **Usuarios:** Una lista que contiene una fecha y dos strings (fecha X string X string). Cabe destacar que, para el caso del usuario activo, solo se guarda en una lista un string.

- **Documentos:** Una lista que contiene una lista que contiene tres strings, una fecha, dos listas y un integer (string X fecha X string X string X list X list X integer)
- **Permisos:** Una lista que contiene como mínimo 2 strings (string X string X.. string)
- **Historial de versiones:** Una lista que contiene una fecha, un string y un integer (fecha X string X integer)

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

- **ParadigmaDocsRegister:** Verifica si un username no ha sido registrado. Si es así, registra al usuario en la plataforma.
- **ParadigmaDocsLogin:** Verifica las credenciales de un usuario registrado y si son correctas, deja al usuario como usuario activo (logueo correcto). Es necesario que un User este logueado para que las operaciones a continuación funcionen correctamente.
- **ParadigmaDocsCreate:** Crea un nuevo documento y lo guarda en la plataforma.
- **ParadigmaDocsShare:** Permite compartir un documento con otros usuarios registrados en la plataforma. Los permisos se guardan en el documento especifico.
- **ParadigmaDocsAdd:** Añade texto al final de la versión activa de un documento.
- **ParadigmaDocsRestoreVersion:** Restaura una versión de un documento, guardada en el historial.
- **ParadigmadocsToString:** Convierte el contenido de Paradigmadocs en un string.
- **ParadigmaDocsRevokeAllAccesses:** Elimina todos los permisos externos otorgados en un documento.
- **ParadigmaDocsSearch:** Permite buscar documentos (propios o que se tenga permisos) que contengan un texto en específico. Retorna una lista con los documentos encontrados.
- **ParadigmaDocsDelete:** Permite eliminar texto en una versión activa de un documento.
- **ParadigmaDocsSearchAndReplace:** Permite buscar y reemplaza un texto en una versión activa de un documento.

Cabe destacar de que todos los predicados desde ParadigmaDocsCreate hasta ParadigmaDocsSearchAndReplace requieren de un login correcto para obtener el resultado esperado, a excepción de ParadigmadocsToString, donde esta operación de igual forma retorna todo el contenido de Paradigmadocs si no hubo un Login exitoso. ParadigmadocsTostring retorna un string que debe ser utilizado a través del predicado “write” para que sea visualizado correctamente.

A diferencia del primer laboratorio, no es necesario que todos los TDAs estén en archivos independientes y por lo tanto se puede tener un solo archivo que contenga los TDAs y los predicados obligatorios y opcionales. Se pide que todos los predicados estén documentados correctamente, señalando dominios, descripción, predicados y metas cuando corresponda.

2.2 DISEÑO DE LA SOLUCIÓN

Para diseñar la solución, no solo es necesario utilizar los tipos de datos nativos de Prolog, sino que también se deben crear tipos de datos específicos, a través de lo que se conoce como TDA, utilizando la siguiente estructura: Representación, Constructores, Predicados de Pertenencia, Selectores, Modificadores y otros predicados asociados al TDA. Sin embargo, cabe destacar de que en Prolog, un predicado puede actuar como muchas cosas a la vez.

Los TDAs creados son utilizados para la construcción de las operaciones de otros TDAs y para las operaciones obligatorias y opcionales. Los TDAs más importantes son:

2.2.1.1 TDA PARADIGMADOCS

- Representación: Una lista que contiene: un string, una fecha, una lista para guardar a los usuarios registrados, una lista para guardar al usuario activo y una lista para guardar a los documentos.
- Constructor Y Pertenencia: Dados un nombre y una fecha, se crea una lista que tiene los elementos anteriormente mencionados junto con tres listas vacías. Se verifica que los elementos correspondan a los tipos de datos correctos.
- Selectores Y Modificadores: *Ver Tabla N°1 en Anexos, P. 13.*

2.2.1.2 TDA USER

- Representación: Una lista que contiene una fecha y dos strings.
- Constructor Y Pertenencia: Dados una fecha de registro, un username y un password, se crea una lista que contiene estos tres elementos. Se verifica que los elementos correspondan a los tipos de datos correctos.
- Selectores Y Otros Predicados: *Ver Tabla N°2 en Anexos, P. 13.*

2.2.1.3 TDA DOCUMENTO

- Representación: Una lista que contiene tres strings, una fecha, dos listas y un integer
- Constructor Y Pertenencia: Dados un autor, una fecha, un nombre, un texto y un ID, se crea una lista que contiene las entradas anteriormente mencionadas junto con las dos listas. La segunda lista corresponde a un Historial de Versiones. Se verifica que los elementos correspondan a los tipos de datos correctos.
- Selectores Y Otros Predicados: *Ver Tabla N°3 en Anexos, P. 14.*

2.2.1.4 OTROS TDAS

Los TDAs más pequeños, pero que se utilizan en otros TDAs son:

- **TDA Fecha:** Representado como una lista que contiene tres integers (DD/MM/AAAA)
- **TDA Historial:** Representado como una lista que contiene una fecha, un string y un integer

Ver la Figura N°1 en ANEXOS para poder tener más información sobre el uso de TDAs en otros TDAs. P.15.

2.2.2 OPERACIONES OBLIGATORIAS

PARADIGMADOCSREGISTER: Se implementa un predicado el cual permite verificar si un User está registrado en la plataforma y si no es así, se registra al User. Si en la última entrada se coloca un paradigmaDocs que contiene al User registrado, se verifica que el paradigmaDocs resultante coincida con el paradigmaDocs ingresado en la última entrada.

PARADIGMADOCSLOGIN: Se implementa un predicado que permite loguear a un User si corresponde a uno de los Users registrados en la plataforma. Para ello, se verifica que el username y password ingresados coincida con los datos de algún Users registrado. Si el login es exitoso, el paradigmaDocs de salida contiene el nombre del User en la lista de User Activo. Los predicados a continuación requieren que haya un user logueado mediante este predicado para que funcionen correctamente.

PARADIGMADOCSCREATE: Se implementa un predicado que crea un Documento cuyo autor corresponde al User activo. A través de un predicado se verifica el largo de la lista Documentos para así asignarle el ID correcto al documento. Se agrega a la lista de documentos de ParadigmaDocs y se elimina la sesión activa del User.

PARADIGMADOCSSHARE: Se implementa un predicado, donde se recibe una lista de usuarios y una lista de permisos. Se verifica que el ID corresponda a uno de los documentos de Paradigmadocs. Se filtran los permisos y que los users existan y luego vuelve a agregar el Documento a paradigmaDocs, esta vez con los permisos agregados y se elimina el documento sin permisos. Se elimina la sesión activa del User.

PARADIGMADOCSADD: Se implementa un predicado donde se verifica que el ID corresponda a uno de los documentos de ParadigmaDocs. A través de otros predicados se verifica que el user activo sea el autor del documento o bien tenga permisos para editar. Se agrega el texto al final de la versión activa, creando una versión nueva y la antigua queda guardada en el historial. Se elimina la sesión activa del User.

PARADIGMADOCSRESTOREVERSION: Se implementa un predicado donde se verifica que el ID corresponda a uno de los documentos de Paradigmadocs. Luego se verifica que el ID del historial corresponda a una de las versiones y se reemplaza la versión activa del Documento por una antigua del Historial. Se elimina la sesión activa del User.

PARADIGMADOCSTOSTRING: Se implementa un predicado donde dependiendo de si hay un User logueado, se transforma la información del User a string (su username, fecha de registro y documentos a los cuales puede acceder). Si no hay User logueado entonces toda la información de paradigmaDocs se transforma a string. Para ello se utiliza string_concat y atomics_to_string junto con otros predicados de los TDAs.

2.2.3 OPERACIONES OPCIONALES

PARADIGMADOCSREVOKEALLACCESES: Se implementa un predicado que recibe una lista de documentos de IDs y a través de esto obtiene los documentos si es que el usuario activo tiene acceso a ellos. Usando maplist, elimina los permisos externos y actualiza a estos documentos en ParadigmaDocs. Se elimina la sesión activa del User.

PARADIGMADOCSSEARCH: Se implementa un predicado que a través del uso del predicado nativo maplist y sub_string busca a todos los documentos que contengan un texto en especifico y que sean del User Activo (propio o que tenga permisos de escritura o lectura). Retorna una lista de documentos donde la búsqueda dio éxito. Se elimina la sesión activa del User.

PARADIGMADOCSDELETE: Se implementa un predicado que a través del uso del predicado nativo sub_string elimina una cantidad de caracteres en específico de la versión activa de un documento. Una versión alternativa del predicado es cuando el numero de caracteres a eliminar es mayor al largo del string de la versión activa, donde en ese caso la versión activa se reemplaza con “”. Se elimina la sesión activa del User.

PARADIGMADOCSSEARCHANDREPLACE: Se implementa un predicado que mezcla los procedimientos de ParadigmaDocsSearch y ParadigmaDocsDelete, donde primero se utiliza sub_string para asegurarse de que el string a reemplazar este en el documento y luego se utiliza re_replace de manera recursiva para ir reemplazando todas las coincidencias de la palabra a reemplazar. Se repite el proceso hasta formar una nueva versión activa del documento. Se elimina la sesión activa del User.

2.3 ASPECTOS DE IMPLEMENTACIÓN

2.3.1 COMPILADOR / IDE

Para este proyecto es necesario el IDE Swi-Prolog, específicamente de versión 8.x.x o superior. Como alternativas, también es posible utilizar Visual Studio Code con la extensión de Swi-Prolog o bien, se puede hacer uso de SWISH (Prolog online).

La implementación se baja en el trabajo con listas, sin embargo, se espera que algunas operaciones de listas como por ejemplo member o cualquiera que no sea append, length o maplists sean implementadas.

2.3.2 ESTRUCTURA DEL CÓDIGO

Como en este laboratorio es posible trabajar todo en un solo archivo, no fue necesario llamar predicados de otros archivos externos al archivo main. No se utilizaron librerías especiales, si no que solo se hizo uso de algunos predicados nativos de prolog, como son string_concat, sub_string, length, append, maplist, not, entre otros.

Al abrir el archivo main, se puede ver que esta organizado en el siguiente orden: TDA Fecha, TDA User, TDA Historial, TDA Documento, TDA ParadigmaDocs, Predicados Obligatorios, Predicados Opcionales y Ejemplos para los predicados obligatorios y opcionales.

2.4 INSTRUCCIONES DE USO

2.4.1 EJEMPLOS DE USO

Lo primero que se debe hacer es ejecutar una instrucción que permitirá visualizar con claridad todas las listas, ya que a veces Prolog acorta las listas para que no se extiendan tanto. Para evitar eso, se coloca el siguiente comando: *set_prolog_flag(answer_write_options,[max_depth(0)]).*

Una vez ejecutado, ahora se puede cargar la base de conocimientos. Simplemente se debe ir a la opción File->Consult y luego seleccionar el archivo "main_20537567_SerranoCarrasco.pl". Si todo sale bien, prolog dará como respuesta "true.", lo cual significa que la base de conocimiento fue cargada con éxito.

Una vez ejecutado, en el archivo main se tienen varios ejemplos para cada una de las operaciones. Solo se debe copiar el ejemplo y pegarlo en la consola de Prolog, asegurándose de que el ejemplo este bien escrito y no se haya copiado el % de comentario. Si todo resulta, Prolog mostrará en la consola las distintas fechas y paradigmaDocs resultantes.

En el caso de querer probar el código sin utilizar los ejemplos anteriormente mencionados, se deben seguir los siguientes pasos y **respetando el siguiente orden**:

1. Crear una o varias Fecha. Esto se hace con el predicado date.
2. Crear un paradigmaDocs. Esto se hace con el predicado paradigmaDocs.
3. Registrar a usuarios en la plataforma. Esto se hace con el predicado paradigmaDocsRegister.
4. Loguear a un usuario en la plataforma. Esto se hace con el predicado paradigmaDocsLogin.
5. Crear un Documento. Esto se hace con el predicado paradigmaDocsCreate.
6. ¡Y listo! Ahora ya se tiene todo para poder probar el programa con normalidad. Para realizar otra operación se debe repetir desde el paso 4 hacia adelante, ya que todas las operaciones a excepción de paradigmaDocsRegister y paradigmaDocsLogin requieren de Users logueados.

Un ejemplo de lo anterior seria: *date(03, 01, 2022, D1),paradigmaDocs("Google docs", D1, PD1),paradigmaDocsRegister(PD1,D1,"user1","pass1",PD2),paradigmaDocsLogin(PD2,"user1","pass1",PD3),paradigmaDocsCreate(PD3,D1,"Doc1","Content1",PD4).*

Ver en Anexos la Figura N°2, la cual es un ejemplo paso a paso con el procedimiento anterior para el caso del predicado paradigmaDocsCreate. P.16. Ver Figuras N°3 – N°5. P.16 - 17 para ver más ejemplos.

2.4.2 RESULTADOS ESPERADOS

Se espera crear una simulación de una plataforma de edición de texto colaborativo, donde el programa sea funcional en su totalidad, sin errores ni ambigüedades.

Junto con lo anterior, se espera trabajar correctamente con listas y recursión, ya que son fundamentales para el desarrollo del proyecto. Por último, se espera que cada predicado haga su procedimiento correctamente y que los TDAs tengan representaciones correctas.

2.4.3 POSIBLES ERRORES

Posibles errores pueden suceder si se utiliza mal el orden de los predicados de las operaciones o bien, si no se utilizan correctamente las variables a la hora de ejecutar el código. Cabe destacar de que estos casos son imprevistos que no están considerados a la hora de realizar el código, pues se tiene en consideración que siempre el usuario realizará correctamente la ejecución del código siguiendo los formatos establecidos en los ejemplos y con el orden correcto (Por ejemplo, usar Login antes de cada operación a excepción de Register, etc).

2.5 RESULTADOS Y AUTOEVALUACIÓN

2.5.1 RESULTADOS OBTENIDOS

Los resultados obtenidos fueron los esperados, ya que se logró crear TODOS los predicados obligatorios y opcionales. El programa es completamente funcional, incluyendo en casos de contraejemplos donde las entradas colocadas son erróneas. Por lo tanto, se logró crear una simulación de una plataforma de edición de texto colaborativo.

Se hicieron múltiples pruebas con distintos ejemplos para probar de que no hubiera fallos en la ejecución del código y que el código hiciera lo correcto.

2.5.2 AUTOEVALUACIÓN

La Autoevaluación se realiza de la siguiente forma: 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.

Debido a que se probaron los predicados con varios ejemplos y no se encontraron errores, se considera que todos los predicados funcionan el 100% de las veces.

Para ver la tabla de Autoevaluación, ver la Tabla N°4 en Anexos, P.17

3. CONCLUSIÓN

Luego de realizar y completar el proyecto, se puede concluir que se cumplieron los objetivos principales, ya que fue posible aprender a utilizar correctamente Prolog para poder completar el proyecto de laboratorio correspondiente. Se obtuvo aprendizaje sobre un paradigma nuevo y se pudo aplicar correctamente, utilizando los conceptos vistos en clase como son los TDAs, recursividad, hechos, reglas, base de conocimiento, entre otros.

Una de las complicaciones más grande para la realización del proyecto, fue que a veces algunas ideas no resultaban bien del todo y posterior a ver el resultado de una idea se decidía cambiarla si es que era necesario. Esto para evitar errores o facilitar la lectura del código, como, por ejemplo, al principio se tenían algunos predicados donde las variables no daban a entender muy bien que elemento o tipo de dato debía ser ingresado, por lo que al final del proyecto se decidió modificar estos aspectos, para facilitar la lectura propia y del corrector. Otra gran complicación era el tiempo, ya que este es un proyecto que requiere de bastante tiempo para su comprensión y para lograr realizar todo lo requerido, pero, aun así, se pudo aprovechar bien e incluso se pudieron realizar todos los predicados tanto obligatorios como opcionales.

Dejando lo anterior de lado, no hubo complicaciones de comprensión de conceptos o problemas con las herramientas como Git o Prolog en general. Al contrario, gracias a este proyecto se pudieron aprender muchas herramientas y predicados nativos de Prolog, algo que ayudó mucho también a la hora de preparar evaluaciones. Se espera que este conocimiento también sea útil para el desarrollo del próximo laboratorio.

Comparado con el paradigma anterior, el cual era Paradigma Funcional, en el paradigma Lógico resultó un código mucho mas corto y simple, gracias al hecho de que muchos predicados pueden ser utilizados para varias cosas distintas, lo que también provoca que se ahorren muchas líneas de código, algo que en Scheme no pasaba debido a que una función tenía un solo propósito en específico. Personalmente, considero que el Paradigma Lógico es mucho mas simple de trabajar comparado con Scheme, especialmente porque esta vez se puede utilizar y emplear el concepto de Variable, algo que es esencial para el paradigma Lógico.

4. BIBLIOGRAFÍA Y REFERENCIAS

1. Flores, V. (2021). “Proyecto Semestral de Laboratorio”. Paradigmas de Programación. Enunciado de Proyecto Online. Recuperado de: https://docs.google.com/document/d/1pORdJwp5WJ7V3DIAQik9B58llpdxpurQRVTKPZHOpS8/edit
2. Flores, V. (2021). “4 - P. Lógico”. Paradigmas de Programación. Material de clases Online. Recuperado de: https://uvirtual.usach.cl/moodle/course/view.php?id=10036&section=16
3. Chacon, S. y Straub, B. (2020). “Pro Git – Todo lo que necesitas saber sobre Git’’. Libro Online. Recuperado de : https://drive.google.com/file/d/1mHJsfvGCYcIhdrmK-lBI6a1WS-U1AAPi/view
4. Multiples Autores. (Septiembre 2013). “El lenguaje de programación PROLOG”. Libro online. Recuperado de: https://drive.google.com/file/d/1DgxI64oAB5do7A-ajCXCTphnGWHTCEmk/view?usp=sharing
5. Autores Desconocidos. (2021). “Swi-Prolog Documentation’’. Swi-Prolog. Documentación Online. Recuperado de: https://www.swi-prolog.org/pldoc/index.html

5. ANEXOS

Tabla N°1: Predicados restantes del TDA Paradigmadocs.

Tipo de predicado	Nombre	Descripción
Selector	getNombrePdocs	Obtiene el nombre de un ParadigmaDocs
Selector	getDatePdocs	Obtiene la fecha de creación de un ParadigmaDocs
Selector	getRegistradosPdocs	Obtiene la lista de Users registrados de ParadigmaDocs
Selector	getActivosPdocs	Obtiene la lista de Users activos de ParadigmaDocs
Selector	getUserActivoPdocs	Obtiene al User activo de la lista de Users activos de ParadigmaDocs
Selector	getDocumentosPdocs	Obtiene la lista de Documentos de ParadigmaDocs
Modificador	paradigmadocsActualizado	Crea un paradigmaDocs actualizado, donde cualquiera de sus elementos puede ser actualizado y/o cambiado.

Tabla N°2: Predicados restantes del TDA User.

Tipo de predicado	Nombre	Descripción
Selector	getUserDate	Obtiene la fecha de registro de un User
Selector	getUserName	Obtiene el Username de un User
Selector	getUserPass	Obtiene el Password de un User
Otros Predicados	pertenece	Verifica si un Username corresponde a alguno de los Users en una lista de Users
Otros Predicados	buscarUsuarioPassword	Verifica si existe un User en una lista de Users tal que el Username y Password coincidan con los ingresados por entradas
Otros Predicados	verificarListaUsuarios	Verifica si todos los Usernames de una lista pertenecen a Users de una lista de Users
Otros Predicados	obtenerDatosUser	Encuentra a un User en específico en una lista de Users y obtiene sus datos e información
Otros Predicados	userToString	Transforma toda la información de un User (incluido el password) a string.

Tabla N°3: Predicados restantes del TDA Documento.

Tipo de función	Nombre	Descripción
Selector	getAutorDocumento	Obtiene el autor de un Documento
Selector	getDateDocumento	Obtiene la fecha de creación de un Documento
Selector	getNameDocumento	Obtiene el nombre de un Documento
Selector	getContenidoDocumento	Obtiene el contenido de un Documento
Selector	getPermisosDocumento	Obtiene la lista de permisos de un Documento
Selector	getHistorialDocumento	Obtiene la lista del Historial de versiones de un Documento
Selector	getIDDocumento	Obtiene el ID de un Documento
Otros Predicados	encontrarPorIDDocumento	Encuentra un documento en especifico a través de su ID en una lista de documentos
Otros Predicados	primerElemento	Obtiene el primer elemento de cualquier lista
Otros Predicados	permisoValido	Verifica si un permiso es de lectura, escritura, compartir o comentarios
Otros Predicados	buscarPermiso	Obtiene (si existe) la lista de permisos de un User en específico en un Documento en especifico
Otros Predicados	verificarCompartir	Verifica si en la lista de permisos de un User se encuentra el permiso de compartir (“S”)
Otros Predicados	puedeCompartir	Recorre todas las listas de permisos de un Documento para encontrar la que esta asociada a un User y verificar si puede compartir
Otros Predicados	verificarEditar	Verifica si en la lista de permisos de un User se encuentra el permiso de escritura (“W”)
Otros Predicados	puedeEditar	Recorre todas las listas de permisos de un Documento para encontrar la que está asociada a un User y verificar si puede editar
Otros Predicados	obtenerListaAccesos	Asigna todos los permisos validos a los usuarios correspondientes y coloca estos permisos en la lista de permisos de un Documento (Sin filtrar)
Otros Predicados	tienePoder	Verifica si un User de una lista de Users tiene algún permiso en un Documento (sin importar el tipo)
Otros Predicados	tienenPermisos	Verifica cuales son los Users que tienen permisos en un Documento y si estos corresponden a los Users de una lista de Users
Otros Predicados	borrarPrimeraOcurrenciaPermiso	Elimina una lista de permisos duplicada de un User. Siempre elimina la lista más antigua.
Otros Predicados	borrarRepetidos	Filtra la lista de permisos de un Documento y elimina aquellas listas repetidas
Otros Predicados	encontrarPorIDHistorial	Encuentra una versión del historial de versiones de un documento, a través de su ID
Otros Predicados	elementoPermiosString	Transforma un elemento de una lista de permisos a string (varía dependiendo de que elemento sea)
Otros Predicados	permisosToString	Transforma una lista de permisos a string
Otros Predicados	fullAccessesToString	Transforma toda una lista de sublistas de permisos a string
Otros Predicados	documentoToString	Transforma todo un Documento a string

Otros Predicados	puedeAcceder	Permite saber si algún User tiene algún permiso
Otros Predicados	obtenerDocsAcceso	Permite obtener todos los documentos donde un User en específico tiene algún permiso
Otros Predicados	searchInDocuments	Predicado que verifica si un texto se encuentra en la versión activa de un documento. Si es así, lo agrega a una lista y verifica los demás documentos
Otros Predicados	searchInDocsHist	Predicado que obtiene todas las versiones de un documento y verifica si alguna de ellas tiene un texto en especifico
Otros Predicados	unirBusquedas	Predicado que une las listas de documentos donde se encontró un texto en especifico
Otros Predicados	reemplazarPalabras	Reemplaza todas las coincidencias de una palabra por otra
Otros Predicados	eliminarTodosLosPermisos	Elimina todos los permisos (excepto del autor) de un documento
Otros Predicados	obtenerDocumentosPropios	Obtiene todos los documentos propios de un autor en especifico de acuerdo con una lista de IDs
Otros Predicados	eliminarDocsDuplicados	Elimina documentos que tienen un mismo ID de una lista de documentos
Otros Predicados	agregarNuevosDocs	Agrega un conjunto de documentos a una lista
Otros Predicados	obtenerDocumentosPorAutor	Obtiene todos los documentos de un Autor en específico.

Figura N°1: Diagrama de uso de cada de TDAs.

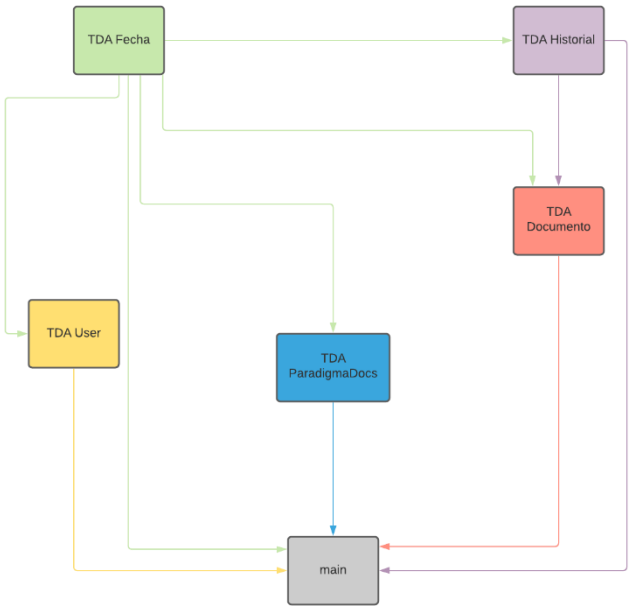
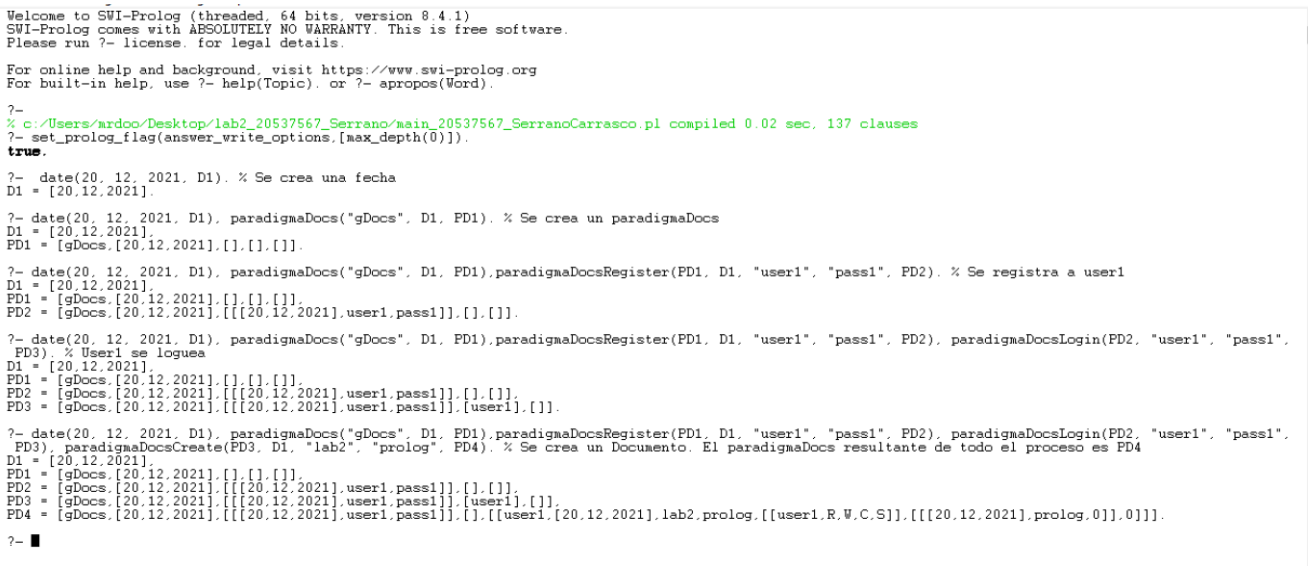


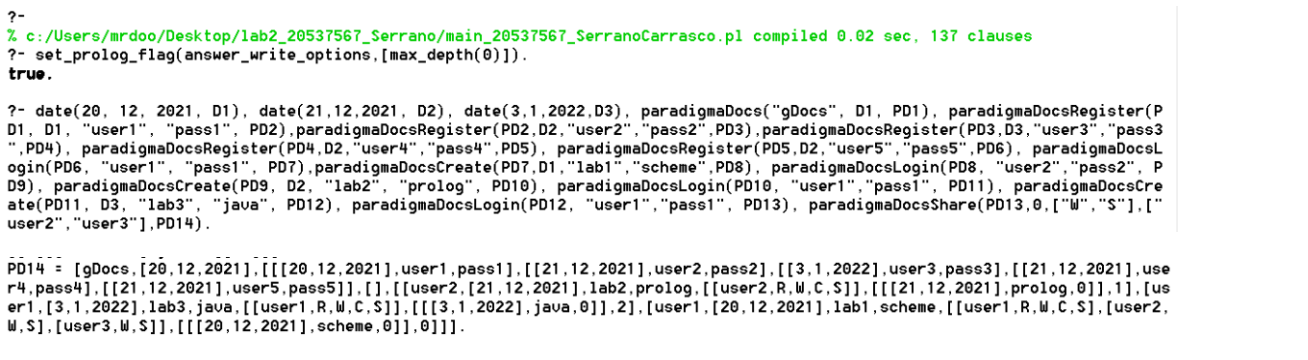
Figura N°2: Ejemplo número 1: Paso a paso para utilizar la función Create.



```
Welcome to SWI-Prolog (threaded, 64 bits, version 8.4.1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit https://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?-
% c:/Users/mrdoo/Desktop/lab2_20537567_Serrano/main_20537567_SerranoCarrasco.pl compiled 0.02 sec, 137 clauses
?- set_prolog_flag(answer_write_options,[max_depth(0)]).
true.

?-  date(20, 12, 2021, D1). % Se crea una fecha
D1 = [20,12,2021].

?- date(20, 12, 2021, D1), paradigmaDocs("gDocs", D1, PD1). % Se crea un paradigmaDocs
D1 = [20,12,2021],
PD1 = [gDocs,[20,12,2021],[],[],[]].

?- date(20, 12, 2021, D1), paradigmaDocs("gDocs", D1, PD1),paradigmaDocsRegister(PD1, D1, "user1", "pass1", PD2). % Se registra a user1
D1 = [20,12,2021],
PD1 = [gDocs,[20,12,2021],[],[],[]],
PD2 = [gDocs,[20,12,2021],[[[20,12,2021],user1,pass1]],[],[]].

?- date(20, 12, 2021, D1), paradigmaDocs("gDocs", D1, PD1),paradigmaDocsRegister(PD1, D1, "user1", "pass1", PD2), paradigmaDocsLogin(PD2, "user1", "pass1",
 PD3). % User1 se loguea
D1 = [20,12,2021],
PD1 = [gDocs,[20,12,2021],[],[],[]],
PD2 = [gDocs,[20,12,2021],[[[20,12,2021],user1,pass1]],[],[]],
PD3 = [gDocs,[20,12,2021],[[[20,12,2021],user1,pass1]],[user1],[]].

?- date(20, 12, 2021, D1), paradigmaDocs("gDocs", D1, PD1),paradigmaDocsRegister(PD1, D1, "user1", "pass1", PD2), paradigmaDocsLogin(PD2, "user1", "pass1",
 PD3), paradigmaDocsCreate(PD3, D1, "lab2", "prolog", PD4). % Se crea un Documento. El paradigmaDocs resultante de todo el proceso es PD4
D1 = [20,12,2021],
PD1 = [gDocs,[20,12,2021],[],[],[]],
PD2 = [gDocs,[20,12,2021],[[[20,12,2021],user1,pass1]],[],[]],
PD3 = [gDocs,[20,12,2021],[[[20,12,2021],user1,pass1]],[user1],[]],
PD4 = [gDocs,[20,12,2021],[[[20,12,2021],user1,pass1]],[],[[user1,[20,12,2021],lab2,prolog,[[user1,R,W,C,S]],[[[20,12,2021],prolog,0]],0]]].

?-
```

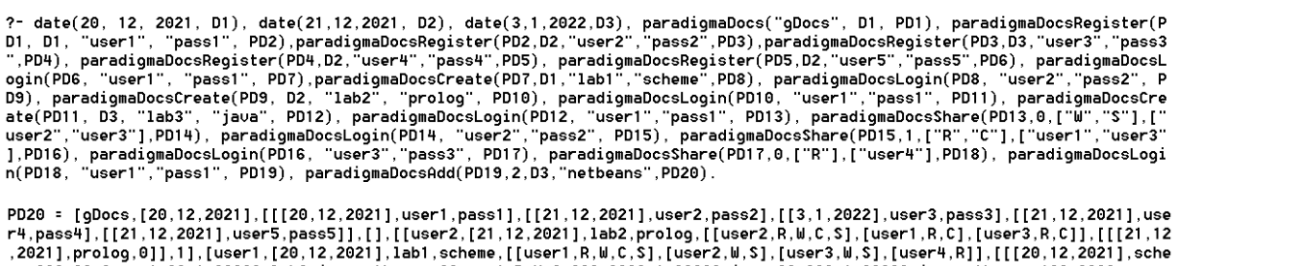
Nota: Al final del archivo main están los demás ejemplos listos para ejecutar. Las siguientes figuras muestran algunos de estos.

Figura N°3: Ejemplo "listo para ejecutar" del predicado paradigmaDocsShare. El resultado final está en la variable PD14.



```
?-
% c:/Users/mrdoo/Desktop/lab2_20537567_Serrano/main_20537567_SerranoCarrasco.pl compiled 0.02 sec, 137 clauses
?- set_prolog_flag(answer_write_options,[max_depth(0)]).
true.

?- date(20, 12, 2021, D1), date(21,12,2021, D2), date(3,1,2022,D3), paradigmaDocs("gDocs", D1, PD1), paradigmaDocsRegister(P
D1, D1, "user1", "pass1", PD2),paradigmaDocsRegister(PD2,D2,"user2","pass2",PD3),paradigmaDocsRegister(PD3,D3,"user3","pass3
",PD4), paradigmaDocsRegister(PD4,D2,"user4","pass4",PD5), paradigmaDocsRegister(PD5,D2,"user5","pass5",PD6), paradigmaDocsL
ogin(PD6, "user1", "pass1", PD7),paradigmaDocsCreate(PD7,D1,"lab1","scheme",PD8), paradigmaDocsLogin(PD8, "user2","pass2", P
D9), paradigmaDocsCreate(PD9, D2, "lab2", "prolog", PD10), paradigmaDocsLogin(PD10, "user1","pass1", PD11), paradigmaDocsCre
ate(PD11, D3, "lab3", "java", PD12), paradigmaDocsLogin(PD12, "user1","pass1", PD13), paradigmaDocsShare(PD13,0,["W","S"],["
user2","user3"],PD14).

PD14 = [gDocs,[20,12,2021],[[[20,12,2021],user1,pass1],[[21,12,2021],user2,pass2],[[3,1,2022],user3,pass3],[[21,12,2021],use
r4,pass4],[[21,12,2021],user5,pass5]],[],[[user2,[21,12,2021],lab2,prolog,[[user2,R,W,C,S]],[[[21,12,2021],prolog,0]],1],[us
er1,[3,1,2022],lab3,java,[[user1,R,W,C,S]],[[[3,1,2022],java,0]],2],[user1,[20,12,2021],lab1,scheme,[[user1,R,W,C,S],[user2,
W,S],[user3,W,S]],[[[20,12,2021],scheme,0]],0]]].
```

Figura N°4: Ejemplo "listo para ejecutar" del predicado paradigmaDocsAdd. El resultado final está en la variable PD20



```
?- date(20, 12, 2021, D1), date(21,12,2021, D2), date(3,1,2022,D3), paradigmaDocs("gDocs", D1, PD1), paradigmaDocsRegister(P
D1, D1, "user1", "pass1", PD2),paradigmaDocsRegister(PD2,D2,"user2","pass2",PD3),paradigmaDocsRegister(PD3,D3,"user3","pass3
",PD4), paradigmaDocsRegister(PD4,D2,"user4","pass4",PD5), paradigmaDocsRegister(PD5,D2,"user5","pass5",PD6), paradigmaDocsL
ogin(PD6, "user1", "pass1", PD7),paradigmaDocsCreate(PD7,D1,"lab1","scheme",PD8), paradigmaDocsLogin(PD8, "user2","pass2", P
D9), paradigmaDocsCreate(PD9, D2, "lab2", "prolog", PD10), paradigmaDocsLogin(PD10, "user1","pass1", PD11), paradigmaDocsCre
ate(PD11, D3, "lab3", "java", PD12), paradigmaDocsLogin(PD12, "user1","pass1", PD13), paradigmaDocsShare(PD13,0,["W","S"],["
user2","user3"],PD14), paradigmaDocsLogin(PD14, "user2","pass2", PD15), paradigmaDocsShare(PD15,1,["R","C"],["user1","user3"
],PD16), paradigmaDocsLogin(PD16, "user3","pass3", PD17), paradigmaDocsShare(PD17,0,["R"],["user4"],PD18), paradigmaDocsLogi
n(PD18, "user1","pass1", PD19), paradigmaDocsAdd(PD19,2,D3,"netbeans",PD20).

PD20 = [gDocs,[20,12,2021],[[[20,12,2021],user1,pass1],[[21,12,2021],user2,pass2],[[3,1,2022],user3,pass3],[[21,12,2021],use
r4,pass4],[[21,12,2021],user5,pass5]],[],[[user2,[21,12,2021],lab2,prolog,[[user2,R,W,C,S],[user1,R,C],[user3,R,C]],[[[21,12
,2021],prolog,0]],1],[user1,[20,12,2021],lab1,scheme,[[user1,R,W,C,S],[user2,W,S],[user3,W,S],[user4,R]],[[[20,12,2021],sche
me,0]],0],[user1,[3,1,2022],lab3,javanetbeans,[[user1,R,W,C,S]],[[[3,1,2022],java,0],[[3,1,2022],javanetbeans,1]],2]]].
```

Figura N°5: Ejemplo "listo para ejecutar" del predicado paradigmaDocsToString. El resultado está en la variable Result y se puede ver cómo queda el string final. (Cabe destacar de que muestra el string en formato correcto debido a que se ingresó el comando mencionado en Instrucciones de Uso).

```
?- date(20, 12, 2021, D1), date(21,12,2021, D2), date(3,1,2022,D3), paradigmaDocs("gDocs", D1, PD1), paradigmaDocsRegister(P
D1, D1, "user1", "pass1", PD2),paradigmaDocsRegister(PD2,D2,"user2","pass2",PD3),paradigmaDocsRegister(PD3,D3,"user3","pass3
",PD4), paradigmaDocsRegister(PD4,D2,"user4","pass4",PD5), paradigmaDocsRegister(PD5,D2,"user5","pass5",PD6), paradigmaDocsL
ogin(PD6, "user1", "pass1", PD7),paradigmaDocsCreate(PD7,D1,"lab1","scheme",PD8), paradigmaDocsLogin(PD8, "user2","pass2", P
D9), paradigmaDocsCreate(PD9, D2, "lab2", "prolog", PD10), paradigmaDocsLogin(PD10, "user1","pass1", PD11), paradigmaDocsCre
ate(PD11, D3, "lab3", "java", PD12), paradigmaDocsLogin(PD12, "user1","pass1", PD13), paradigmaDocsShare(PD13,0,["W","S"],["
user2","user3"],PD14), paradigmaDocsLogin(PD14, "user2","pass2", PD15), paradigmaDocsShare(PD15,1,["R","C"],["user1","user3"
],PD16), paradigmaDocsLogin(PD16, "user3","pass3", PD17), paradigmaDocsShare(PD17,0,["R"],["user4"],PD18), paradigmaDocsLogi
n(PD18, "user1","pass1", PD19), paradigmaDocsAdd(PD19,2,D3,"netbeans",PD20), paradigmaDocsLogin(PD20, "user3","pass3", PD21)
, paradigmaDocsAdd(PD21, 0, D2, "racket", PD22), paradigmaDocsLogin(PD22, "user1", "pass1", PD23), paradigmaDocsToString(PD2
3, Result).

Result = ----------
Nombre de Usuario: user1
Fecha de registro: 20/12/2021
Los documentos propios del usuario o los que tiene acceso son:
----------
ID de Documento: 1
Autor: user2
Fecha de creacion: 21/12/2021
Contenido (Version actual) del Documento: prolog
Permisos:
user2: tiene permisos de lectura - tiene permisos de escritura - tiene permisos de comentarios -  tiene permisos de comparti
r -
user1: tiene permisos de lectura - tiene permisos de comentarios -
user3: tiene permisos de lectura - tiene permisos de comentarios -
Historial de Versiones:
Version Numero 0: * prolog * version guardada en la fecha: 21/12/2021
----------
----------
ID de Documento: 2
Autor: user1
Fecha de creacion: 3/1/2022
Contenido (Version actual) del Documento: javanetbeans
Permisos:
user1: tiene permisos de lectura - tiene permisos de escritura - tiene permisos de comentarios -  tiene permisos de comparti
r -
Historial de Versiones:
Version Numero 0: * java * version guardada en la fecha: 3/1/2022
Version Numero 1: * javanetbeans * version guardada en la fecha: 3/1/2022
----------
----------
```

Tabla N°4: Autoevaluación de los requerimientos funcionales

Requerimientos Funcionales	Evaluación
TDAs	1
ParadigmaDocsRegister	1
ParadigmaDocsLogin	1
ParadigmaDocsCreate	1
ParadigmaDocsShare	1
ParadigmaDocsAdd	1
ParadigmaDocsRestoreVersion	1
ParadigmaDocsToString	1
ParadigmaDocsRevokeAllAccesses	1
ParadigmaDocsSearch	1
ParadigmaDocsDelete	1
ParadigmaDocsSearchAndReplace	1