UNIVERSIDAD
DE SANTIAGO
DE CHILE

Informe
Paradigmas de Programación
Laboratorio 2:
BIBLIOTECA VIRTUAL

Estudiante: Benjamín Goycolea Contardo
Rut: 21.310.769-0
Profesora: Vic Flores Sanchez
Fecha de entrega: 7 de Diciembre de 2025

1. Introducción

En este informe se describe la implementación de un sistema de gestión de biblioteca virtual, cuyo propósito es modelar y automatizar las operaciones de una biblioteca.

La problemática a abordar es implementar la simulación de las operaciones fundamentales de una biblioteca moderna, incluyendo la gestión de usuarios, el catálogo de libros, préstamos, devoluciones, y la aplicación de reglas dinámicas.

Este sistema se implementó en el paradigma lógico, que a su vez es parte del paradigma declarativo. En el paradigma se enfoca en "que" debería hacer el programa, no el "cómo", en el paradigma lógico la respuesta al "que" es la definición de relaciones lógicas entre hechos y reglas.

El informe se estructura presentando el problema y el paradigma, seguido por análisis de requisitos, diseño de la solución con sus algoritmos, aspectos de implementación, instrucciones de uso, resultados y autoevaluación, concluyendo con un contraste con el paradigma funcional del laboratorio anterior.

2. Descripción del Problema

El problema consiste en diseñar y construir un sistema de gestión de biblioteca virtual que permita:

- **Registrar usuarios** (id, nombre, deuda, estado de suspensión, lista de libros en posesión), y gestionarlos.
- **Registrar y mantener un catálogo de libros** (id, título, autor, disponibilidad).
- **Gestionar préstamos y devoluciones**, calcular multas por días de retraso y aplicar suspensión cuando corresponda.
- **Hacer consultas a la biblioteca** sobre usuarios, libros y préstamos.
- **Procesar automáticamente** un día completo.

3. Paradigma Lógico

El paradigma declarativo es un paradigma de programación, véase, una forma o estilo de pensar cómo escribir un programa, en el que se expresa la lógica de lo que este debe realizar, sin describir un control de flujo predeterminado, como lo es en el paradigma iterativo, o sea, se describe el qué debe hacer un programa, no el cómo.
En relación a esto el paradigma lógico es un paradigma de programación basado en la lógica matemática formal, donde el programa consiste en un conjunto de

hechos y reglas que definen relaciones entre objetos, y la ejecución es un proceso de inferencia lógica para responder consultas.
Este paradigma se basa en la lógica de predicados, lo que nos lleva a sus principales propiedades aplicadas en el proyecto:

- **Hechos y Reglas:** Los hechos son declaraciones verdaderas que describen relaciones entre objetos. Las reglas definen nuevas relaciones en función de otras.
- **Unificación:** El mecanismo mediante el cual Prolog encuentra valores para las variables que satisfacen una consulta, emparejando términos.
- **Backtracking:** Prolog explora automáticamente diferentes posibilidades cuando una solución falla, retrocediendo y probando alternativas hasta encontrar una solución válida o agotar las opciones.
- **Predicados:** Son las unidades básicas de código que establecen relaciones lógicas. Pueden tener múltiples cláusulas que se evalúan de arriba hacia abajo.
- **Recursión:** Similar al paradigma funcional, se utiliza recursión en lugar de iteración para procesar listas y estructuras de datos.
- **Pattern Matching:** Las cláusulas de los predicados se seleccionan mediante coincidencia de patrones, lo que permite una forma natural de descomponer estructuras de datos.

4. Análisis del Problema

El problema más a fondo es el conjunto de reglas o requisito que debe cumplir la biblioteca:

1. **Creación e Inicio del Sistema:** Este requisito implica en primer lugar crear algo que sea el “sistema” o biblioteca, que más allá de poder crearse con los parámetros iniciales solicitados, implica crear una entidad que además almacene el resto de los elementos que conforman la biblioteca en sí, los libros, usuarios, y los préstamos.
2. **Gestión de Usuarios:** Este requisito implica tanto abstraer y crear a los usuarios en sí, así como poder gestionarlos, o sea crear funciones que permitan administrar el estado actual de los usuarios.
3. **Gestión de Libros:** Hay que representar y crear una abstracción de los libros que se puedan mantener almacenados en el sistema. Se debe poder agregar nuevos libros al sistema, por lo que se debe crear una función que añade nuevos libros al sistema, pero asegurarse de que no tengan el ID duplicado. A su vez hay que crear funciones de búsqueda de libros.
4. **Préstamos y Devoluciones:** En esta parte hay que crear una función para efectuar un préstamo, que debe verificar varias condiciones tanto para el usuario como para el libro en sí.

Después, al autorizar el préstamo, hay que hacer actualizaciones por varias partes, tanto por el usuario, el libro, así como crear el préstamo en sí.
Finalmente hay que crear una función para poder devolver el libro, que si bien no se restringe la devolución, si ocurren distintos eventos en caso de ciertas condiciones u otras. Similar a la función para efectuar un préstamo.

5. **Multas y Deudas:** Esto va ligado al punto anterior, que, si un libro no es devuelto a tiempo, además de realizar los procedimientos normales, también hay que darle una multa al usuario en base a cuanto se demoro en devolver por sobre el tiempo límite, y en caso de superar una deuda limite, suspenderlo.
6. **Suspensión de Usuarios:** Esta parte tiene implicancias en los puntos anteriores, como que si está suspendido no puede solicitar un libro. Para este punto hay que crear una función que permita al usuario pagar total o parcialmente su deuda, y que en caso de que no deba más libros, levantar la suspensión.
7. **Procesamiento del Tiempo:** Esta parte es para poder avanzar un día. Hay que realizar una función que realice todos los cambios que ocurren de un día a otro, considerando que ya no hay más cambios externos. Actualizar la fecha, los tiempos de cada préstamo, en caso de haberse pasado del tiempo empezar a sumar deuda, entre otras.
8. **Consultas y Reportes:** Para cumplir con este punto hay que crear funciones que permitan buscar por libros en base a distintos parámetros, e imprimir su información.

5. Diseño de la Solución

El diseño se basa en la creación de Tipos de Datos Abstractos (TDA) para cada entidad principal, utilizando listas de Prolog como estructura de datos subyacente.

Diseño de TDAs:
Cada TDA (libro, usuario, prestamo, biblioteca) se implementó con:

- **Constructor:** Un predicado "crear<TDA>" que recibe los atributos iniciales y unifica una instancia del TDA (una lista) con el último parámetro.
- **Selectores:** Predicados "get<TDA><Atributo>" que extraen un valor específico de la instancia del TDA mediante unificación del segundo parámetro.
- **Modificadores:** Predicados "set<TDA><Atributo>" que reciben una instancia, un nuevo valor, y unifican una nueva instancia con el valor actualizado en el tercer parámetro, manteniendo la inmutabilidad. Además predicados auxiliares "agregar<Elemento>Usuario" y "remover<Elemento>Usuario" que, para los atributos que son una lista, agregan o remueven entradas.

Enfoque de Solución y Algoritmos:

- **Estado Inmutable:** El estado global del sistema está contenido en el TDA "biblioteca". Cada predicado que modifica el estado, como "tomarPrestamo" o "procesarDia", recibe una "biblioteca" de entrada y unifica una "biblioteca" nueva con el último parámetro. Con esta forma de modificar el estado actual se evita el uso de una biblioteca mutable.
- **Composición de Predicados:** Las operaciones complejas se descomponen en predicados más pequeños. Por ejemplo, "tomarPrestamo" es una secuencia de validaciones mediante conjunciones lógicas. Si todas las validaciones son exitosas (todas las cláusulas se satisfacen), se invoca a un predicado auxiliar "realizarPrestamo" que compone los modificadores necesarios para actualizar el estado. Si alguna validación falla, el predicado completo falla y no se modifica la biblioteca.
- **Uso de Predicados de Lista:** Para procesar listas y buscar elementos, se usaron predicados estándar de Prolog:
  - "member/2" se usó para verificar pertenencia y buscar elementos en listas.
  - "maplist/2" y "maplist/3" se usaron para aplicar predicados a todos los elementos de una lista.
  - "findall/3" y "filter" (implementados con recursión) se usaron para recolectar elementos que cumplen un criterio.
- **Pattern Matching y Unificación:** Se aprovechó extensivamente el pattern matching de Prolog para descomponer estructuras de datos. Por ejemplo, los selectores de TDA como "getLibroId([Id, _, _, _], Id)" utilizan pattern matching directo en la cabeza del predicado, lo que hace el código conciso y declarativo.
- **Recursión y Backtracking:** Para operaciones como "procesarUsuarios" que debe aplicar verificaciones a todos los usuarios, se utilizó recursión con caso base (lista vacía) y caso recursivo. El backtracking natural de Prolog permite que predicados como "buscarLibro" encuentren la primera coincidencia automáticamente, y el uso del cut (!) previene búsquedas adicionales cuando se encuentra la solución deseada.

6. Aspectos de Implementación

- **Lenguaje e Intérprete:** El proyecto fue desarrollado en SWI-Prolog, utilizando el intérprete swipl versión 8.4 o superior.

- **Estructura del Proyecto:** El código se organizó en archivos modulares para separar las responsabilidades de cada TDA, mejorando la legibilidad y el mantenimiento.
  - **"libro_21310769_GoycoleaContardo.pl":** TDA para libros con predicados de búsqueda.
  - **"usuario_21310769_GoycoleaContardo.pl":** TDA para usuarios

con gestión de deuda y suspensión.
- **"prestamo_21310769_GoycoleaContardo.pl":** TDA para préstamos y predicados de manejo de fechas.
- **"biblioteca_21310769_GoycoleaContardo.pl":** TDA principal que integra las demás y contiene la lógica de negocio.
- **"main_21310769_GoycoleaContardo.pl":** Archivo que utiliza "use_module" para importar y exportar todos los predicados necesarios, actuando como una fachada pública del sistema.
- **"script_base_21310769_GoycoleaContardo.pl":** Script de prueba básico independiente.
- **"script_pruebas_21310769_GoycoleaContardo.pl":** Script de pruebas completo que valida todos los RF.

- **Bibliotecas:** No se emplearon bibliotecas externas, únicamente los predicados estándar provistos por SWI-Prolog.

7. Instrucciones de Uso

Para ejecutar el programa, es necesario tener SWI-Prolog instalado. Las pruebas se pueden ejecutar desde la línea de comandos, situándose en el directorio "Codigo Fuente/".

Ejecutar la prueba básica: Este script simula un flujo de operaciones simple (préstamo, retraso, devolución con multa, pago y suspensión).

bash (Linux o MacOS), cmd (Windows)

```
swipl script_base_21310769_GoycoleaContardo.pl
```

- **Resultado Esperado:** Se mostrará en la consola una traza de las operaciones realizadas día a día, indicando el estado del sistema, las deudas y las suspensiones.

Ejecutar la prueba completa: Este script ejecuta una batería de pruebas que valida cada uno de los 26 Requerimientos Funcionales (RF) de forma individual.

bash (Linux o MacOS), cmd (Windows)

```
swipl script1_21310769_GoycoleaContardo.pl
```

- **Resultado Esperado:** Se imprimirá una salida detallada para cada RF, mostrando que el predicado correspondiente se comporta como se espera.

Posibles Errores:

- **"ERROR: `[Thread main] `<archivo>' does not exist":** Este error ocurrirá si el comando se ejecuta desde un directorio incorrecto. Asegúrese de estar en la carpeta "Codigo Fuente".
- **"ERROR: Exported procedure <predicado> is not defined":** Este error indica que falta la implementación de un predicado exportado. Si está llamando predicados manualmente asegúrese de que escribió bien el nombre de este, y de haber cargado el archivo main.

8. Resultados y Autoevaluación

Resumen de Resultados: Se implementó el 100% de los 26 requerimientos funcionales (RF01 a RF26) especificados en el enunciado.

Pruebas Realizadas: Se crearon dos scripts de prueba principales:
- **"script_base_21310769_GoycoleaContardo.pl":** Un script de integración que prueba una secuencia lógica de eventos.
- **"script_pruebas_21310769_GoycoleaContardo.pl":** Un conjunto de pruebas más unitarias que verifica cada RF de forma aislada.

Todas las pruebas se ejecutaron sin error, lo que muestra que funcionan tanto aisladas unas de otras, como así en cadena, con predicados usando la unificación de otros.

9. Conclusiones del Trabajo

El paradigma lógico demostró ser efectivo para construir un sistema basado en reglas complejas. La naturaleza declarativa de Prolog permitió expresar las restricciones del negocio de forma natural mediante predicados que se leen casi como especificaciones. El patrón matching y la unificación simplifican significativamente la manipulación de estructuras de datos comparado con lenguajes imperativos.

Sin embargo, el paradigma lógico también presentó desafíos. La inmutabilidad, aunque beneficiosa para la predictibilidad, requiere crear nuevas instancias completas de la biblioteca en cada operación, lo que puede ser ineficiente en sistemas grandes.

Comparando con el paradigma funcional (Laboratorio 1): Ambos paradigmas comparten la naturaleza declarativa y la inmutabilidad. Sin embargo, Prolog ofrece mayor expresividad para definir relaciones complejas y realizar búsquedas, mientras que Racket proporciona mejor control sobre la composición de operaciones y funciones de orden superior. En términos de legibilidad, considero que el código funcional es ligeramente más intuitivo para operaciones secuenciales, donde se aplica una función por sobre el retorno de otra. Pero esto

lleva rápidamente a encadenamientos grandes cuando se realizan operaciones complejas, mientras que Prolog produce un código más elegante.

En definitiva, el paradigma lógico es particularmente útil para sistemas basados en reglas y restricciones como el que se implementó, aunque para aplicaciones con operaciones altamente secuenciales o que requieren eficiencia computacional, otros paradigmas podrían ser más apropiados en un escenario real. La elección del paradigma debe considerar la naturaleza del problema a resolver.

10. Referencias

Ai, T. A. (2025, 30 enero). *An Introduction to Prolog*. Applied AI Blog.

https://www.appliedaicourse.com/blog/introduction-to-prolog/

CAMPUS VIRTUAL: Iniciar sesión en el sitio. (s. f.).

https://uvirtual.usach.cl/moodle/course/view.php?id=10036§ion=20

GeeksforGeeks. (2025, 11 julio). *Prolog | An Introduction*. GeeksforGeeks.

https://www.geeksforgeeks.org/artificial-intelligence/prolog-an-introduction/

manual. (s. f.).

https://www.swi-prolog.org/pldoc/doc_for?object=manual

Prolog - Introduction. (s. f.).

https://www.tutorialspoint.com/prolog/prolog_introduction.htm

Una introducción a Prolog. (s. f.). Universidad de Sevilla.

https://www.cs.us.es/~fsancho/Blog/posts/Introduccion_Prolog.md.html