Laboratorio 2
Paradigmas de Programación

Programación Lógica- Prolog
Facultad de Ingeniería- Universidad De Santiago De Chile

Estudiante: Renato Cruz
Profesor: Victor Flores

Índice

1. Introducción

La gestión de recursos en un sistema de biblioteca implica el manejo de múltiples entidades interrelacionadas, tales como usuarios, libros, fechas de vencimiento y estados financieros (multas). La complejidad de estas reglas de negocio —donde una condición (deuda) bloquea una acción (préstamo)— hace que este problema sea un candidato ideal para ser abordado mediante la Programación Lógica.

2. Descripción breve del problema

Se requiere desarrollar un sistema que administre el inventario y los préstamos de una biblioteca. El problema central radica en el control de restricciones temporales y de estado:

- Los libros tienen un stock limitado y no pueden prestarse si no están disponibles.
- Los préstamos tienen una fecha de vencimiento fija.
- El paso del tiempo debe generar multas automáticas si el libro no se ha devuelto.
- El sistema debe impedir nuevos préstamos a usuarios que tengan multas impagas o que hayan excedido el límite de libros permitidos.

3. Descripción breve del Paradigma

El proyecto utiliza el paradigma de **Programación Lógica**. A diferencia de la programación imperativa (que describe *cómo* hacer las cosas paso a paso), la programación lógica se centra en describir *qué* es el problema mediante hechos y reglas.

- **Hechos:** Verdades absolutas del sistema .
- **Reglas:** Relaciones condicionales que infieren nueva información (ej. "Un usuario está suspendido SI su deuda es mayor a 0").

4. Análisis del problema

Para modelar la solución, se identificaron las siguientes entidades y restricciones:

- **Entidades:**
    - *Biblioteca:* El contenedor del estado global (listas de libros, usuarios y préstamos).
    - *Usuario:* Posee un historial de préstamos y un estado financiero (deuda).
    - *Libro:* Recurso compartido con disponibilidad finita.

    - Prestamo: Estructura que relaciona usuarios y libros
    - *Fecha:* Estructura necesaria para calcular días transcurridos y vencimientos.
- **Paso del tiempo:** el paso del tiempo se simula mediante una función de transición (`procesarDia`) que actualiza el estado de la biblioteca, incrementando las multas de los préstamos vencidos día a día.
- **Restricciones de Integridad:** Un usuario no puede prestar el mismo libro dos veces simultáneamente, ni superar el cupo máximo definido al crear la biblioteca.

5. Diseño de solución

La arquitectura se basa en la manipulación del estado a través de predicados que reciben una biblioteca de entrada (`Bin`) y generan una nueva biblioteca de salida (`Bout`).

- **Estructura de Datos:** Se utilizaron listas para almacenar colecciones y estructuras compuestas (functores) para los objetos.
    - `biblioteca(Libros, Usuarios, Prestamos, Config...)`
    - `usuario(ID, Nombre, Estado, Deuda)`
- **Lógica de Negocio:**
    - `tomarPrestamo`: Verifica disponibilidad, cupo del usuario y estado de suspensión antes de agregar un nuevo préstamo a la lista.
    - `devolverLibro`: Busca el préstamo activo, calcula la multa final (si aplica) y actualiza la deuda del usuario.
    - `procesarDia`: Recorre recursivamente todos los préstamos activos. Si la fecha actual supera la fecha de vencimiento, aplica la multa configurada a la deuda del usuario correspondiente.

6. Aspectos de implementación

El sistema fue implementado en **SWI-Prolog**. Los aspectos técnicos más destacados incluyen:

- **Recursividad:** Utilizada extensivamente para recorrer las listas de usuarios y préstamos (ej. al buscar un libro por ID o actualizar la deuda de un usuario específico)
- **Inmutabilidad:** Dado que las variables en Prolog son inmutables, cada cambio de estado (como un nuevo préstamo) resulta en la creación de una nueva estructura de biblioteca que se pasa al siguiente predicado.
- **Manejo de Negación:** Uso de `not/1` o `\+/1` para validar que un libro *no* esté prestado antes de asignarlo.

7. Instrucciones de uso

1. **Carga:** Abrir la terminal de SWI-Prolog y cargar el archivo principal: `consult('biblioteca.pl').`.
2. **Inicialización:** Se puede utilizar el script de prueba incluido copiando el script de demostración.
3. **Consultas Manuales:**
    - Crear biblioteca: `crearBiblioteca([...], B1).`
    - Realizar préstamo: `tomarPrestamo(B_Actual, IdUser, IdLibro, Dias, Fecha, B_Nueva).`
    - Verificar estado: `obtenerUsuario(B_Nueva, IdUser, U), isUsuarioSuspendido(U).`

8. Resultados y Autoevaluación

- **Resultados:** Las pruebas de escritorio (script de integración) demostraron que el sistema maneja correctamente el ciclo de vida del préstamo. Se verificó exitosamente que:
    - Los usuarios acumulan deuda día a día tras el vencimiento.
    - El sistema bloquea préstamos a usuarios morosos (Caso de prueba "Fola suspendida").
    - El pago total de la deuda reactiva inmediatamente al usuario.
- **Autoevaluación:** La lógica implementada es robusta para los requerimientos planteados. Como limitación, el sistema actual no persiste los datos en disco (base de datos), por lo que la información se pierde al cerrar la sesión de Prolog, lo cual es esperado en este ámbito académico.

9. Conclusiones del trabajo

El desarrollo de este sistema permitió comprender la potencia de la programación declarativa para modelar reglas de negocio estrictas. La implementación de la lógica de suspensión y multas resultó ser mucho más limpia y directa en Prolog que en lenguajes funcionales, gracias a la facilidad para definir reglas como `isUsuarioSuspendido`. El mayor desafío fue la gestión del "estado" de la biblioteca a través de las llamadas recursivas, lo cual se resolvió satisfactoriamente mediante el paso de parámetros acumuladores (`Acumuladores`).

10. Referencias

1. Documentación oficial de SWI-Prolog. Disponible en: https://www.swi-prolog.org/
2. Material docente del curso de Paradigmas de Programación (2025).