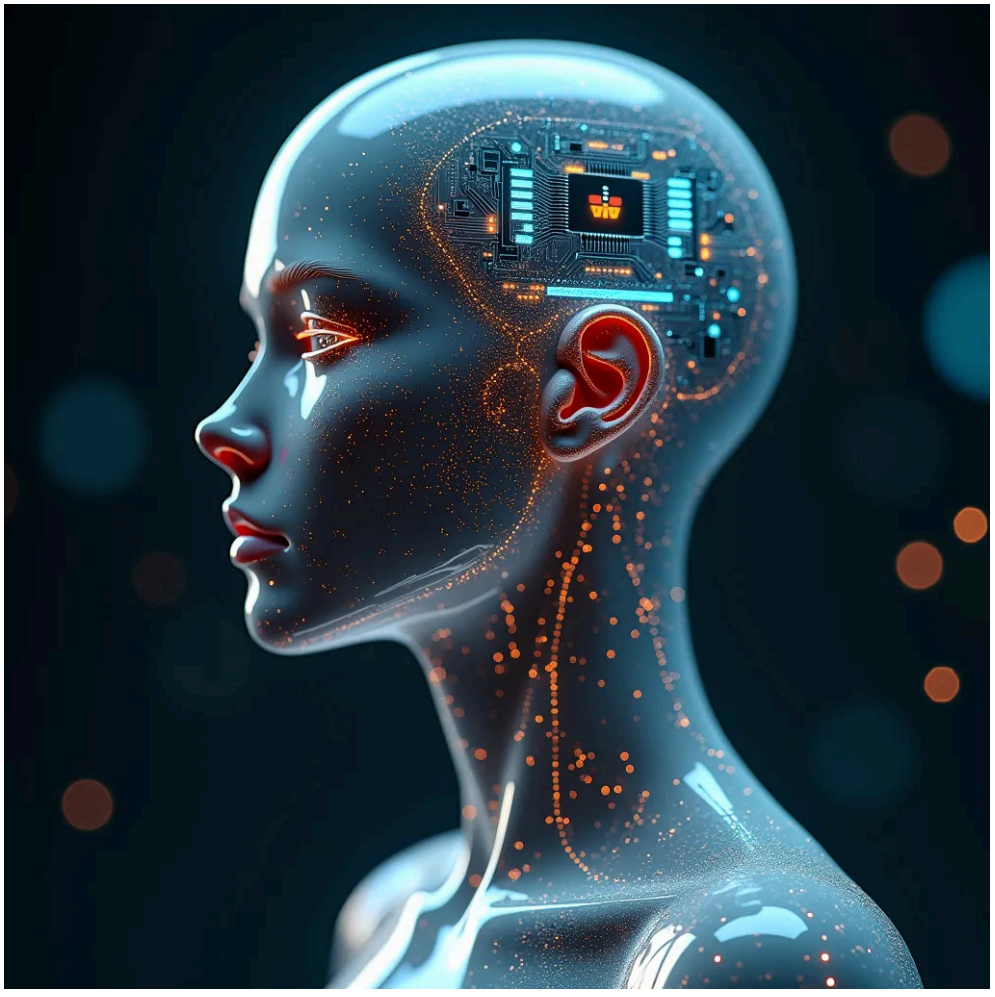
UD02 Práctica01: Sistema Experto con Prolog.



1. Introducción a Prolog

Prolog (acrónimo de *Programming in Logic*) es un lenguaje de programación declarativo diseñado en la década de 1970, basado en la lógica de primer orden y en el concepto de relaciones entre hechos y reglas. A diferencia de los lenguajes de programación imperativos, donde el programador debe describir detalladamente los pasos a seguir para resolver un problema, en Prolog el programador define qué es cierto (hechos) y cómo se relacionan las cosas (reglas), y el motor de inferencia del lenguaje se encarga de encontrar soluciones a partir de las consultas hechas sobre esas reglas y hechos.

Prolog es ampliamente utilizado en áreas como:

- **Inteligencia Artificial**: debido a su facilidad para manejar reglas lógicas y bases de conocimiento.
- **Procesamiento del Lenguaje Natural**: ya que se adapta bien a la manipulación de estructuras complejas de datos.
- **Sistemas expertos**: por su capacidad de modelar y razonar sobre el conocimiento.

1.1. Estructura Básica de Prolog

Prolog se basa en tres componentes fundamentales:

1. **Hechos**: Expresan conocimiento simple que se asume como verdadero. Ejemplo:

```
es_padre(juan, maria).
```

   Aquí se afirma que "Juan es el padre de María".

2. **Reglas**: Describen relaciones lógicas que dependen de otros hechos. Ejemplo:

```
es_abuelo(X, Z) :- es_padre(X, Y), es_padre(Y, Z).
```

   Esta regla dice que "X es abuelo de Z si X es padre de Y, y Y es padre de Z".

3. **Consultas**: El usuario puede hacer preguntas al sistema. Ejemplo:

```
?- es_padre(juan, maria).
```

Esta consulta pregunta si "Juan es padre de María".

El motor de Prolog utiliza un mecanismo de inferencia basado en la **resolución de Horn** y **backtracking** (retroceso) para intentar satisfacer las consultas.

2. Introducción a SWI-Prolog

SWI-Prolog es una implementación gratuita y de código abierto del lenguaje Prolog. Es ampliamente utilizada en investigación, docencia y desarrollo comercial debido a su robustez, extensibilidad y su capacidad de integrar múltiples librerías y herramientas.

2.1. Características principales de SWI-Prolog:

- **Interactividad**: SWI-Prolog ofrece un entorno interactivo que permite ejecutar consultas y cargar programas de manera rápida.
- **Extensiones**: Soporta muchas extensiones al estándar ISO de Prolog, incluyendo interfaces gráficas (XPCE), comunicación con bases de datos, multihilo, sockets de red, y soporte para la web semántica.
- **Multiplataforma**: Funciona en sistemas operativos como Linux, Windows y macOS.

2.2. Instalación por defecto de SWI-Prolog

Para instalar SWI-Prolog, puedes visitar la página oficial SWI-Prolog.org y seguir las instrucciones según tu sistema operativo. También puedes instalarlo utilizando un administrador de paquetes como `apt` en Linux o `brew` en macOS.

En Linux:

```
sudo apt-get install swi-prolog
```

En macOS con Homebrew:

```
brew install swi-prolog
```

En Windows, puedes descargar un instalador desde el sitio web oficial.

2.3. Instalación en Linux con PPA

Estos PPA se actualizan con cada nueva versión. Gracias a Yves Raimond por configurar el PPA y Eugeniy Meshcheryakov por crear la configuración de Debian. El PPA se registra mediante apt-add-repository, que está disponible de forma predeterminada en los escritorios, pero no en los servidores o contenedores de Linux.

- Abre la terminal de Linux.
- Se instala utilizando el comando:

```
% sudo apt-get install software-properties-common
```

- Y después ingrese los siguientes comandos para versiones estables:

```
% sudo apt-add-repository ppa:swi-prolog/stable
% sudo apt-get update
% sudo apt-get install swi-prolog
```

2.4. Uso Básico de SWI-Prolog

1. **Iniciar SWI-Prolog**: Después de la instalación, puedes abrir el intérprete de Prolog en la terminal escribiendo `swipl`.

   ```
   $ swipl
   Welcome to SWI-Prolog (threaded, 64 bits, version X.Y.Z)
   ?-
   ```

   Esto abre un entorno interactivo donde puedes empezar a hacer consultas o cargar programas.

2. **Cargar un archivo**: Puedes escribir tus hechos y reglas en un archivo `.pl` y luego cargarlo en el entorno interactivo con el comando:

   ```
   ?- consult('archivo.pl').
   ```

3. **Hacer consultas**: Una vez cargado el archivo, puedes realizar consultas basadas en los hechos y reglas definidos.

4. **Salir**: Para salir de SWI-Prolog, usa el comando:

```
?- halt.
```

2.4.1. Ejemplo simple en SWI-Prolog

Supongamos que tenemos un archivo `familia.pl` con los siguientes hechos y reglas:

```
% Hechos
es_padre(juan, maria).
es_padre(pedro, juan).
es_padre(juan, carlos).

% Regla
es_abuelo(X, Y) :- es_padre(X, Z), es_padre(Z, Y).
```

Puedes cargar el archivo y hacer consultas como:

```
?- es_abuelo(pedro, maria).
true.

?- es_padre(juan, carlos).
true.
```

2.5. Documentación de SWI Prolog.

- Para mayor información: Sitio oficial

3. Ejemplo de sistema experto en Prolog

```
% Sistema experto para diagnosticar por qué un ordenador no se enciende.

% Hechos iniciales (desconocidos)
:- dynamic(suministro_electrico/1).
:- dynamic(fuente_alimentacion/1).
:- dynamic(boton_power/1).
:- dynamic(cable_video/1).

% Reglas para diagnosticar el problema
diagnostico :-
    comprobar_suministro,
    comprobar_fuente,
    comprobar_boton,
    comprobar_cable,
    write('Todo parece estar en orden, el ordenador debería encender.'), nl.

diagnostico :-
    write('Revisa el diagnóstico de los componentes.'), nl.

% Reglas para comprobar cada componente
comprobar_suministro :-
    suministro_electrico(esta_bien), !,
    write('El suministro eléctrico está bien.'), nl.
comprobar_suministro :-
    write('Comprueba el suministro eléctrico, parece que está fallando.'), nl,
fail.

comprobar_fuente :-
    fuente_alimentacion(esta_bien), !,
    write('La fuente de alimentación está funcionando correctamente.'), nl.
comprobar_fuente :-
    write('La fuente de alimentación podría estar fallando.'), nl, fail.

comprobar_boton :-
    boton_power(funciona), !,
    write('El botón de encendido está bien.'), nl.
comprobar_boton :-
    write('El botón de encendido no está funcionando correctamente.'), nl, fail.

comprobar_cable :-
```

```
    cable_video(conectado), !,
    write('El cable de vídeo está correctamente conectado.'), nl.
comprobar_cable :-
    write('El cable de vídeo no está conectado o está defectuoso.'), nl, fail.

% Preguntas interactivas para conocer el estado de cada componente
main :-
    write(''),nl,
    write('Sistema experto para evaluar el funcionamiento de un ordenador.'), nl,
    write('-----------------------------------------------------------.'), nl,
    preguntar_suministro,
    preguntar_fuente,
    preguntar_boton,
    preguntar_cable,
    diagnostico.

preguntar_suministro :-
    write('¿El suministro eléctrico está bien? (si/no): '),
    read(Respuesta),
    (Respuesta == si -> assertz(suministro_electrico(esta_bien));
     assertz(suministro_electrico(no_funciona))).

preguntar_fuente :-
    write('¿La fuente de alimentación está funcionando correctamente? (si/no):
'),
    read(Respuesta),
    (Respuesta == si -> assertz(fuente_alimentacion(esta_bien));
     assertz(fuente_alimentacion(no_funciona))).

preguntar_boton :-
    write('¿El botón de encendido está funcionando? (si/no): '),
    read(Respuesta),
    (Respuesta == si -> assertz(boton_power(funciona));
     assertz(boton_power(no_funciona))).

preguntar_cable :-
    write('¿El cable de vídeo del monitor está conectado correctamente? (si/no):
'),
    read(Respuesta),
    (Respuesta == si -> assertz(cable_video(conectado));
     assertz(cable_video(desconectado))).
```

¿Cómo funciona el sistema experto?

```
    Hechos dinámicos: Se usa :- dynamic/1 para declarar que los hechos como el estado del suministro eléctrico, la fuente de alimentación, el botón de encendido y el cable de vídeo pueden cambiarse dinámicamente.

    Reglas de diagnóstico: Se definen reglas para comprobar paso a paso cada componente del ordenador. Si todos los componentes están bien, el sistema concluirá que "todo debería estar en orden". De lo contrario, se indicará cuál componente puede estar causando el problema.

    Interactividad: El sistema hace preguntas al usuario sobre el estado de cada componente, y según las respuestas (si o no), se actualizan los hechos y se toma una decisión sobre cuál es el posible fallo.

    Ejecución del sistema: Para comenzar con el diagnóstico, se debe llamar al predicado main/0, que interactúa con el usuario y, al final, da una recomendación basada en las respuestas.
```

Ejemplo de ejecución:

Al ejecutar el sistema, tendrás una interacción similar a esta:

```
?- main.

Sistema experto para evaluar el funcionamiento de un ordenador
--------------------------------------------------------------
¿El suministro eléctrico está bien? (si/no): si.
¿La fuente de alimentación está funcionando correctamente? (si/no): no.
¿El botón de encendido está funcionando? (si/no): si.
¿El cable de vídeo del monitor está conectado correctamente? (si/no): si.
La fuente de alimentación podría estar fallando.
No se ha podido determinar la causa exacta, pero revisa los componentes.
true.
```

En este ejemplo, se indica que la fuente de alimentación es probablemente el problema, pero el sistema también revisa los demás componentes. Si todos los componentes estuvieran en orden, el sistema indicaría que el ordenador debería funcionar correctamente.

Con este enfoque, puedes expandir el sistema agregando más detalles o comprobaciones adicionales según sea necesario.

4. Propuesta

Basándose en el ejemplo y lo presentado en la presente práctica elabora un sistema experto con las siguientes características:

- El sistema realizará un análisis de una situación utilizando para ello un conjunto de preguntas.
- Define los hechos dinámicos y las reglas de diagnóstico de la situación en la que el sistema experto debe actuar.
- Establece las preguntas interactivas que van a modificar los hechos dinámicos.