Informe del Tutorial de Prolog

Tabla de Contenidos

Introducción a Prolog

Prolog, que significa Programacion , es un lenguaje de programación lógico y declarativo. Es particularmente adecuado para aplicaciones en inteligencia artificial y lingüística computacional debido a sus fortalezas en la computación simbólica y no numérica. Los programas en Prolog consisten en una serie de hechos y reglas que describen relaciones y lógica, que el motor de Prolog utiliza para resolver consultas deduciendo nueva información a partir de estas declaraciones.

Configuración del Entorno

Para configurar el entorno de Prolog, el tutorial recomienda utilizar GNU Prolog, versión 1.4.5. La instalación implica descargar el archivo ejecutable desde el sitio web oficial de GNU Prolog, ejecutar el instalador y seguir las instrucciones en pantalla para completar la configuración.

Programa "Hello World"

Un programa simple de "Hello World" en Prolog se puede escribir así:

```
write('Hello World').
```

Este comando se puede ejecutar directamente en la consola de Prolog o guardar en un archivo con la extensión `.pl` y ejecutar desde la consola cargando el .

Conceptos Básicos de Prolog

Hechos, Reglas y Consultas

- **Hechos**: Declaraciones que siempre son verdaderas, como `gato(tom).` que indica que Tom es un gato.
- **Reglas**: Declaraciones condicionales que derivan nuevos hechos. Por ejemplo:

  ```
  abuelo(X, Y) :- padre(X, Z), padre(Z, Y).
  ```

  Esta regla implica que X es abuelo de Y si X es padre de Z y Z es padre de Y.
- **Consultas**: Preguntas que se hacen al sistema Prolog para obtener información basada en los hechos y reglas definidas. Por ejemplo:

  ```
  ?- abuelo(juan, ana).
  ```

  Esto pregunta si Juan es abuelo de Ana.

Codigo de ejemplo

```
% Hechos
padre(juan, maria).
padre(maria, ana).
padre(pedro, juan).

% Regla
abuelo(X, Y) :- padre(X, Z), padre(Z, Y).

% Consulta
```

```
% ?- abuelo(pedro, ana).
% Esto devolverá true ya que Pedro es abuelo de Ana.
```

Relaciones en Prolog

Las relaciones en Prolog se definen mediante predicados que describen cómo se relacionan distintos objetos. Un ejemplo clásico es un árbol genealógico, donde se pueden definir relaciones como `padre(X, Y)` para indicar que X es el padre de Y. Estas relaciones permiten construir y consultar estructuras complejas de datos. Ejemplo:

```
padre(juan, maria).
padre(maria, ana).
```

Esto define que Juan es padre de María y María es madre de Ana.

Codigo de ejemplo

```
% Hechos
madre(maria, jose).
madre(maria, luis).
padre(jose, carla).

% Relaciones
hijo(X, Y) :- padre(Y, X).
hijo(X, Y) :- madre(Y, X).

% Consultas
% ?- hijo(jose, maria).
% true
```

Objetos de Datos

Prolog maneja varios tipos de objetos de datos, incluidos:

- **Átomos**: Constantes simbólicas como `a`, `b`, `tom`.
- **Números**: Valores numéricos.
- **Variables**: Letras mayúsculas o combinaciones de ellas, como `X`, `Y`, `Variable`.
- **Estructuras**: Términos complejos que agrupan varios elementos, como `punto(10, 25)`.

Codigo de ejemplo

```
% Definición de estructura
punto(X, Y).

% Ejemplo de uso
```

```
distancia(punto(X1, Y1), punto(X2, Y2), D) :-
    D is sqrt((X2 - X1)^2 + (Y2 - Y1)^2).

% Consulta
% ?- distancia(punto(1, 1), punto(4, 5), D).
% D = 5.0
```

Operadores

Prolog incluye una variedad de operadores para realizar operaciones aritméticas, comparaciones y manipulación lógica. Algunos ejemplos son:

- **Aritméticos**: `+`, `-`, `*`, `/`
- **Comparación**: `=:=`, `=\=`, `>`, `<`
- **Lógicos**: `,` (AND), `;` (OR), `\+` (NOT)

Ejemplo de uso

```
% Operadores aritméticos
X is 3 + 2.  % X = 5

% Comparación
?- 3 > 2.    % true

% Operadores lógicos
?- (2 < 3, 4 > 1).  % true

?- (2 > 3; 4 > 1).  % true
```

Bucles y Toma de Decisiones

Prolog no tiene estructuras de bucles tradicionales como en otros lenguajes de programación. En su lugar, se utilizan técnicas de recursión para lograr un comportamiento similar. Las decisiones se toman mediante el uso de reglas condicionales y cortes (`!`) para controlar el flujo de la ejecución y prevenir retrocesos innecesarios. Ejemplo:

```
factorial(0, 1).
factorial(N, Result) :-
    N > 0,
    N1 is N - 1,
    factorial(N1, R1),
    Result is N * R1.
```

Este código calcula el factorial de un número utilizando recursión.

Codigo de ejemplo

```
% Cálculo del máximo común divisor (MCD) usando recursión
mcd(X, 0, X) :- !.
mcd(X, Y, D) :-
    Y > 0,
    Resto is X mod Y,
    mcd(Y, Resto, D).

% Consulta
% ?- mcd(48, 18, D).
% D = 6
```

Conjunciones y Disyunciones

Las conjunciones (AND) y disyunciones (OR) en Prolog se implementan usando los operadores `,` y `;` respectivamente. Esto permite combinar múltiples condiciones en una sola regla para evaluar más de un predicado simultáneamente. Ejemplo:

```
es_adulto(Persona) :-
    edad(Persona, Edad),
    Edad >= 18.

puede_votar(Persona) :-
    es_adulto(Persona),
    ciudadano(Persona).
```

Este código verifica si una persona es adulta y puede votar.

Codigo de ejemplo

```
% Hechos
edad(juan, 25).
ciudadano(juan).
edad(ana, 16).

% Reglas
es_adulto(Persona) :-
    edad(Persona, Edad),
    Edad >= 18.

puede_votar(Persona) :-
    es_adulto(Persona),
    ciudadano(Persona).

% Consultas
% ?- puede_votar(juan).
% true
```

```
% ?- puede_votar(ana).
% false
```

Listas en Prolog

Las listas son una estructura de datos fundamental en Prolog. Se utilizan para almacenar secuencias de elementos y se manipulan mediante coincidencia de patrones y recursión. Ejemplo de operaciones con listas:

```
miembro(X, [X|_]).
miembro(X, [_|Resto]) :- miembro(X, Resto).

concatenar([], L, L).
concatenar([X|L1], L2, [X|L3]) :-
    concatenar(L1, L2, L3).
```

Recursión y Estructuras

La recursión es una técnica esencial en Prolog para definir relaciones repetitivas y estructurales. Se utiliza comúnmente para procesar listas, definir árboles y otras estructuras de datos complejas. Un ejemplo clásico es el cálculo del factorial:

```
% Definición del factorial usando recursión
factorial(0, 1). % Caso base
factorial(N, Result) :-
    N > 0,
    N1 is N - 1,
    factorial(N1, R1),
    Result is N * R1.
```

En este ejemplo, `factorial/2` define la relación entre un número y su factorial. El caso base es que el factorial de 0 es 1. Para cualquier otro número `N`, se reduce `N` por 1 y se multiplica por el resultado del factorial de `N-1`.

Codigo de ejemplo

```
% Hechos
padre(juan, maria).
padre(maria, ana).

% Regla recursiva para encontrar ancestros
ancestro(X, Y) :-
    padre(X, Y).
ancestro(X, Y) :-
```

```
    padre(X, Z),
    ancestro(Z, Y).

% Consulta
% ?- ancestro(juan, ana).
% true
```

Retroceso (Backtracking)

El retroceso es el mecanismo mediante el cual Prolog busca todas las posibles soluciones a una consulta. Si una rama de búsqueda falla, Prolog retrocede y prueba alternativas. Este proceso es automático y una parte fundamental de la ejecución de consultas en Prolog.

Ejemplo de Retroceso

```
% Definición de hechos
color(rojo).
color(verde).
color(azul).

% Consulta con retroceso
% ?- color(X).
% X = rojo ;
% X = verde ;
% X = azul ;
% false.
```

En este ejemplo, Prolog prueba todas las posibles soluciones para `X` hasta que se agotan las opciones.

Diferente y No

Prolog ofrece operadores para manejar la negación y la desigualdad. `\+` se utiliza para la negación (not), y `\=` se utiliza para expresar que dos términos son diferentes.

Ejemplo de Negación y Desigualdad

```
% Hechos
hombre(juan).
mujer(maria).

% Negación
es_hombre(X) :- hombre(X).
es_hombre(X) :- \+ mujer(X).

% Desigualdad
diferente(X, Y) :- X \= Y.
```

```
% Consultas
% ?- es_hombre(maria).
% false.

% ?- diferente(juan, maria).
% true.
```

Entradas y Salidas

Prolog proporciona predicados para realizar operaciones de entrada y salida, como leer y escribir en la consola.

Ejemplo de Entrada y Salida

```
% Escribir en la consola
escribir_saludo :- write('Hola, mundo!').

% Leer de la consola
leer_nombre :-
    write('Ingrese su nombre: '),
    read(Nombre),
    write('Hola, '),
    write(Nombre),
    write('!').

% Consultas
% ?- escribir_saludo.
% Hola, mundo!

% ?- leer_nombre.
% Ingrese su nombre: 'Pedro'.
% Hola, Pedro!
```

Predicados Incorporados

Prolog tiene una variedad de predicados incorporados que facilitan operaciones comunes, como el manejo de listas, aritmética y control de flujo.

Ejemplo de Predicados Incorporados

```
% Predicado para sumar elementos de una lista
sumar_lista([], 0).
sumar_lista([Cabeza|Cola], Suma) :-
    sumar_lista(Cola, SumaCola),
    Suma is Cabeza + SumaCola.

% Predicado para encontrar el elemento máximo en una lista
```

```
maximo([X], X).
maximo([X|Xs], Max) :-
    maximo(Xs, MaxTail),
    Max is max(X, MaxTail).

% Consultas
% ?- sumar_lista([1, 2, 3, 4], S).
% S = 10.

% ?- maximo([3, 5, 2, 8, 6], M).
% M = 8.
```

Estructura de Datos en Árbol

Los árboles son estructuras de datos importantes en Prolog, utilizados para representar jerarquías y relaciones anidadas. Los árboles binarios, por ejemplo, se pueden definir y manipular de manera recursiva.

Ejemplo de Árbol Binario

```
% Definición de un árbol binario
arbol(vacio).
arbol(nodo(Izquierda, Raiz, Derecha)).

% Predicado para insertar un elemento en un árbol binario de búsqueda
insertar(X, vacio, nodo(vacio, X, vacio)).
insertar(X, nodo(I, Y, D), nodo(I1, Y, D)) :-
    X =< Y,
    insertar(X, I, I1).
insertar(X, nodo(I, Y, D), nodo(I, Y, D1)) :-
    X > Y,
    insertar(X, D, D1).

% Consulta
% ?- insertar(5, vacio, T), insertar(3, T, T1), insertar(7, T1, T2).
% T2 = nodo(nodo(vacio, 3, vacio), 5, nodo(vacio, 7, vacio)).
```

En este ejemplo, se define un árbol binario y un predicado para insertar elementos en él. Se muestran consultas para construir un árbol binario con los valores 5, 3 y 7.
