+++ date = '2025-05-27T20:20:56-07:00' draft = true title = 'Practica4' +++

Tabla de contenido

Introducción

En el desarrollo de esta práctica, exploraremos el paradigma lógico y el lenguaje de programación Prolog, aprendiendo conceptos y como es que este lenguaje funciona, asi como comprender lo básico. El objetivo es familiarizarnos con los conceptos fundamentales de Prolog, su sintaxis y cómo se utiliza para resolver problemas lógicos.

¿Qué es el Paradigma Lógico?

El paradigma lógico se basa en la idea de que los programas son conjuntos de declaraciones lógicas (hechos y reglas). En lugar de describir *cómo* realizar una tarea (como en los lenguajes imperativos), se describe *qué es verdad* y se deja que el sistema deduzca cómo llegar a esas conclusiones. Un programa lógico consiste en una base de conocimiento, sobre la cual se pueden realizar preguntas (consultas), y el sistema (motor de inferencia) buscará respuestas.

¿Qué es Prolog?

Prolog (PROgramming in LOGic) es el lenguaje de programación más conocido del paradigma lógico. Se basa en la lógica de primer orden y se utiliza ampliamente en inteligencia artificial, procesamiento del lenguaje natural y sistemas expertos. Su motor de inferencia utiliza un algoritmo de resolución para deducir nuevos hechos.

Primeros Pasos con Prolog

Instalación

La implementación más popular de Prolog es SWI-Prolog.

1. **Descargar SWI-Prolog**: Ve al sitio web oficial de SWI-Prolog y descarga la versión para tu sistema operativo.
2. **Instalar SWI-Prolog**: Sigue las instrucciones de instalación.
3. **Verificar**: Abre una terminal y escribe `swipl`. Si ves el prompt `?-`, la instalación fue exitosa.

Alternativamente, puedes usar entornos en línea como SWISH.

"¡Hola, Mundo!" en Prolog

En la consola de Prolog (`?-`), puedes escribir:

```
?- write('¡Hola, Mundo!'), nl.
```

Esto imprimirá "¡Hola, Mundo!" seguido de una nueva línea (`nl`). El sistema responderá `true.` indicando que la consulta fue exitosa. Todas las sentencias en Prolog terminan con un punto (`.`).

Para salir del intérprete, escribe `halt.` o presiona `Ctrl+D` (o `Ctrl+C` y luego `e` para exit).

Es más común escribir código en archivos `.pl` (e.g., `hola.pl`):

```
% hola.pl
saludar :-
```

```
    write('¡Hola desde un archivo!'),
    nl.
```

Luego, en `swipl`, carga el archivo:

```
?- ['hola.pl']. % o consult('hola.pl').
true.

?- saludar.
¡Hola desde un archivo!
true.
```

Comentarios

- Comentario de una línea: `% Esto es un comentario`
- Comentario multilínea: `/* Esto es un comentario ... de varias líneas */`

Conceptos Fundamentales de Prolog

Hechos (Facts)

Los hechos son afirmaciones que se consideran verdaderas sobre el mundo. Declaran relaciones entre objetos.

Guía para escribir hechos

- Los nombres de relaciones (predicados) y objetos (átomos) usualmente empiezan con minúscula.
- La relación aparece primero, seguida de los objetos (argumentos) entre paréntesis, separados por comas.
- Todo hecho debe terminar con un punto (`.`).
- Ejemplo: `gato(tom).` se denomina predicado o cláusula.

Sintaxis de hechos

```
relacion(objeto1, objeto2, ...).
```

Ejemplos de hechos en Prolog

```
gato(tom).
le_gusta_comer(jorge, pasta).
color_pelo(ana, negro).
es_perezoso(juan).
```

Reglas (Rules)

Las reglas permiten definir nuevas relaciones o propiedades basadas en hechos u otras reglas existentes. Una regla establece que algo es verdadero *si* se cumplen ciertas condiciones.

Sintaxis de reglas

```
cabeza(Obj1, Obj2) :- cuerpo_condicion1(Obj1, X), cuerpo_condicion2(X, Obj2).
```

- `cabeza` es la conclusión.
- `:-` se lee como "si".
- `cuerpo` son las condiciones, separadas por comas (`,` que significa "Y" lógico) o punto y coma (`;` que significa "O" lógico).

Si tenemos `P :- Q;R.`, es equivalente a:

```
P :- Q.
P :- R.
```

Si tenemos `P :- Q,R;S,T,U.`, se interpreta como `P :- (Q,R);(S,T,U).`

Ejemplos de reglas en Prolog

```
% Lili está feliz si Lili baila.
feliz(lili) :- baila(lili).

% Tom tiene hambre si Tom busca comida.
hambriento(tom) :- busca_comida(tom).

% Jack y Bili son amigos si a ambos les encanta el cricket.
amigos(jack, bili) :- le_gusta_cricket(jack), le_gusta_cricket(bili).
```

Consultas (Queries)

Las consultas son preguntas que se hacen a la base de conocimiento. Prolog intentará satisfacer la consulta buscando hechos y aplicando reglas.

```
?- gato(tom).        % ¿Es Tom un gato?
true.

?- le_gusta_comer(jorge, pizza). % ¿A Jorge le gusta comer pizza?
false. % (si no está en la base de conocimiento)

?- feliz(lili).      % ¿Está Lili feliz?
% Prolog intentará satisfacer baila(lili).
```

Las variables (empiezan con Mayúscula) permiten hacer preguntas más generales:

```
?- le_gusta_comer(jorge, Comida).
Comida = pasta.
```

Base de Conocimiento

Un conjunto de hechos y reglas forma una base de conocimiento. Usualmente se guarda en un archivo con extensión `.pl`.

Ejemplo: `kb1.pl`

```
% kb1.pl
chica(priya).
chica(natasha).
chica(jasmin).

sabe_cocinar(priya).
```

Para usarla:

```
?- [kb1].
true.

?- chica(X).
X = priya ; % presiona ; para buscar más soluciones
X = natasha ;
X = jasmin.

?- sabe_cocinar(jasmin).
false.
```

Relaciones en Prolog

Prolog se destaca en definir y consultar relaciones complejas.

Ejemplo Práctico: Familia

Considera el siguiente archivo `familia.pl`:

```
% Hechos: parent(PadreMadre, HijoHija)
parent(pam, bob).
parent(tom, bob).
parent(tom, liz).
parent(bob, ann).
parent(bob, pat).
```

```
parent(pat, jim).
parent(peter, jim). % Corregido de 'pete' en PDF para consistencia visual

% Hechos: male(Persona), female(Persona)
male(tom).
male(bob).
male(jim).
male(peter).
female(pam).
female(liz).
female(ann).
female(pat).

% Reglas
mother(Madre, HijoHija) :-
    parent(Madre, HijoHija),
    female(Madre).

father(Padre, HijoHija) :-
    parent(Padre, HijoHija),
    male(Padre).

has_child(Persona) :-
    parent(Persona, _). % _ es una variable anónima

sister(Hermana, Persona) :-
    parent(PadreMadre, Hermana),
    parent(PadreMadre, Persona),
    female(Hermana),
    Hermana \== Persona. % \== significa "no es idéntico a"

brother(Hermano, Persona) :-
    parent(PadreMadre, Hermano),
    parent(PadreMadre, Persona),
    male(Hermano),
    Hermano \== Persona.

grandparent(AbueloAbuela, NietoNieta) :-
    parent(AbueloAbuela, PadreMadreIntermedio),
    parent(PadreMadreIntermedio, NietoNieta).
```

Consultas:

```
?- [familia].
true.

?- mother(X, bob).
X = pam.

?- sister(ann, pat).
true.
```

```
?- grandparent(tom, ann).
true.
```

Recursión en Relaciones Familiares

Podemos definir un "predecesor" (ancestro) recursivamente:

```
% X es predecesor de Z si X es padre/madre de Z (caso base)
predecessor(X, Z) :- parent(X, Z).

% X es predecesor de Z si X es padre/madre de Y,
% y Y es predecesor de Z (caso recursivo)
predecessor(X, Z) :- parent(X, Y), predecessor(Y, Z).
```

Consultas:

```
?- predecessor(tom, jim).
true.
```

Objetos de Datos en Prolog

Prolog maneja varios tipos de datos, llamados términos.

Átomos

Nombres simbólicos. Comienzan con minúscula o están entre comillas simples si contienen espacios o empiezan con mayúscula. Ej: `tom`, `rojo`, `'Hola Mundo'`, `x_45`.

Números

Enteros y de punto flotante. Ej: `100`, `1235`, `3.1416`, `-50`.

Variables

Comienzan con una letra mayúscula o un guion bajo (_). Representan valores desconocidos que Prolog intenta instanciar. Ej: `X`, `Y`, `Nombre`, `_variable`.

Variables Anónimas

Un solo guion bajo (_). Se usa cuando el valor de una variable no es importante. Cada `_` es una variable distinta. Ej: `has_child(X) :- parent(X, _).` (no nos importa quién es el hijo, solo que X tenga uno).

Estructuras

Objetos complejos compuestos por un functor (nombre) y argumentos. Ej: `punto(10, 20)`, `fecha(9, mayo, 2017)`. Los hechos y reglas son en sí estructuras. `parent(pam, bob)` es una estructura con functor `parent` y aridad 2.

Operadores

Operadores de Comparación

Evalúan relaciones numéricas.

Operador	Descripción
`X > Y`	X es mayor que Y
`X < Y`	X es menor que Y
`X >= Y`	X es mayor o igual que Y
`X =< Y`	X es menor o igual que Y
`X =:= Y`	los valores de X e Y son numéricamente iguales
`X =\= Y`	los valores de X e Y son numéricamente diferentes

Ejemplo: `?- 5 > 3.` devuelve `true.`.

Operadores Aritméticos

Para realizar cálculos. La evaluación se hace con el operador `is`.

Operador	Descripción
`+`	Suma
`-`	Resta
`*`	Multiplicación
`/`	División
`**`	Potencia
`//`	División entera
`mod`	Módulo

Ejemplo:

```
?- X is 5 + 3.
X = 8.

?- Y is 10 / 3.
Y = 3.3333333333333335.

?- Z is 10 // 3.
Z = 3.
```

Control de Flujo

Prolog no tiene estructuras de bucle `for` o `while` como otros lenguajes. El control se logra mediante recursión y el mecanismo de backtracking.

Recursión para "Bucles"

Para repetir acciones, se usa la recursión.

```
% Contar desde N hasta Max
contar(Max, Max) :- write(Max), nl.
contar(N, Max) :-
    N < Max,
    write(N), nl,
    N1 is N + 1,
    contar(N1, Max).
```

Consulta: `?- contar(1, 5).` imprimirá 1, 2, 3, 4, 5.

Decisiones mediante Cláusulas

Se pueden simular condicionales usando diferentes cláusulas para un mismo predicado. Prolog intentará unificarlas en orden.

```
% option.pl
comparar(X, Y) :- X > Y, write('X es mayor que Y'), nl.
comparar(X, Y) :- X < Y, write('X es menor que Y'), nl.
comparar(X, Y) :- X =:= Y, write('X e Y son iguales'), nl.
```

Consulta:

```
?- comparar(5, 2).
X es mayor que Y
true.

?- comparar(2, 5).
X es menor que Y
true.
```

Listas en Prolog

Las listas son una estructura de datos fundamental y muy usada.

Representación de Listas

- Lista vacía: `[]`
- Lista con elementos: `[elemento1, elemento2, elemento3]`
- Una lista se compone de:

    - **Cabeza (Head)**: el primer elemento.
    - **Cola (Tail)**: el resto de la lista (que es también una lista).
- El separador `|` se usa para distinguir cabeza y cola: `[Cabeza | Cola]` Ejemplos: `[a, b, c]` es `[a | [b, c]]` `[a, b, c]` es `[a | [b | [c | []]]]` `[a]` es `[a | []]`

Operaciones Básicas con Listas

Comprobación de Pertenencia (member)

Verifica si un elemento está en una lista.

```
% list_basics.pl
miembro(X, [X | _Cola]). % X es miembro si es la cabeza
miembro(X, [_Cabeza | Cola]) :- miembro(X, Cola). % X es miembro si está en la
cola
```

Consulta:

```
?- miembro(b, [a, b, c]).
true.

?- miembro(d, [a, b, c]).
false.
```

Concatenación (append)

Une dos listas.

```
% list_basics.pl
concatenar([], Lista2, Lista2).
concatenar([Cabeza | Cola1], Lista2, [Cabeza | ColaResultado]) :-
    concatenar(Cola1, Lista2, ColaResultado).
```

Consulta:

```
?- concatenar([1, 2], [3, 4], Resultado).
Resultado = [1, 2, 3, 4].
```

Cálculo de Longitud (length)

Encuentra el número de elementos en una lista.

```
% list_basics.pl
longitud([], 0).
longitud([_ | Cola], N) :-
    longitud(Cola, N1),
    N is N1 + 1.
```

Consulta:

```
?- longitud([a, b, c], L).
L = 3.
```

(Nota: `member/2`, `append/3`, `length/2` suelen ser predicados predefinidos en SWI-Prolog).

Retroceso (Backtracking)

Cuando Prolog intenta satisfacer una consulta, puede encontrar múltiples caminos. Si un camino falla, Prolog *retrocede* al último punto de elección e intenta una alternativa. Esto permite encontrar todas las soluciones posibles.

En el intérprete, después de una solución, puedes teclear `;` (punto y coma) para pedirle a Prolog que busque más soluciones.

```
?- miembro(X, [1, 2, 3]).
X = 1 ;
X = 2 ;
X = 3.
```

Para ver el proceso de backtracking, puedes usar `trace.` (y `notrace.` para desactivarlo):

```
?- trace.
true.

[trace]  ?- miembro(X, [a,b]).
   Call: (10) miembro(_11346, [a, b]) ? creep
   Exit: (10) miembro(a, [a, b]) ? creep  % Primera solución: X = a
X = a ;                                   % Pedimos más
   Redo: (10) miembro(a, [a, b]) ? creep
   Call: (11) miembro(_11346, [b]) ? creep
   Exit: (11) miembro(b, [b]) ? creep    % Segunda solución: X = b
X = b ;
   Redo: (11) miembro(b, [b]) ? creep
   Call: (12) miembro(_11346, []) ? creep
   Fail: (12) miembro(_11346, []) ? creep % No hay más
false.
```

Predicados Integrados Útiles

Prolog ofrece muchos predicados predefinidos. Algunos comunes son:

Predicado	Descripción
`var(X)`	Verdadero si X es una variable no instanciada.
`nonvar(X)`	Verdadero si X no es una variable.
`atom(X)`	Verdadero si X es un átomo.
`number(X)`	Verdadero si X es un número.
`integer(X)`	Verdadero si X es un entero.
`float(X)`	Verdadero si X es un número de punto flotante.
`atomic(X)`	Verdadero si X es un átomo o un número.
`compound(X)`	Verdadero si X es un término compuesto (estructura).
`ground(X)`	Verdadero si X no contiene variables no instanciadas.
`write(Term)`	Escribe `Term` en la salida.
`nl`	Escribe una nueva línea.
`read(Term)`	Lee un término de la entrada y lo unifica con `Term`.
`consult(File)`	Carga un archivo Prolog. Similar a `[File].`
`halt.`	Termina la sesión de Prolog.

Conclusión

Esta práctica ha proporcionado una introducción sólida al lenguaje Prolog y al paradigma lógico. Pudimos explorar sus conceptos fundamentales, desde hechos y reglas hasta listas y retroceso. Prolog es una herramienta muy poderosa para resolver problemas lógicos y de inteligencia artificial, y su enfoque declarativo ofrece una perspectiva única en comparación con los lenguajes imperativos tradicionales.

Referencias

- SWI-Prolog Quickstart
- Prolog Tutorial