Prolog

@17 de abril de 2024

Paradigma lógico

La deducción lógica Es una forma de computación universal. Su lenguaje más representativo es Prolog, basado en el princiío de resolución-SL de Kowalski y Kuehner, una regla de inferencia para demostrar que una clausula es consecuencia de un conjunto de ellas y computar los valores de las variables en la cláusula que hacen esto posible

Se utiliza para demostrar la velocidad de una consulta.

Prolog

- Paradigma lógico
- Declarativo
- Resolución de teoremas mediante aplicación de reglas lógicas
- Motor de inferencia (resolución SLD)
- Interpretado
- Tipificación dinámica
- Evaluación ansiosa

En la resolución SLD se busca un camino de resolución la consulta siguiendo una estrategia lineal de selección de cláusulas definida. Esta estrategia se basa en la selección de la primera cláusula que unifique con la consulta y luego sigue de forma lineal las cláusulas que unifican con las variables existentes en la consulta.

En la evaluación ansiosa, los algoritmos de una función o predicado se evalúan completamente antes de que la función o predicado se llame

Se cumple siempre y cuando una variable cuenta con una característica.

Diferencias contra java, c

- No se expresa cómo hacer las cosas sino qué se debe hacer
- Se trabaja a nivel de lógica de primer orden en lugar de lógica preposicional
- Los programas no se ejecutan, se le pregunta al interprete.
- A las variables no se les asigna un valor, éstas unifican un valor
- Muchas de las clausulas de control tradicionales no están (o al menos no son necesarias) Esto incluye aunque no se limita a: if, while, for, or
- La negación tiene un significado especial (negación de mundo cerrado)
- Las estructuras de datos son diferentes (no hay arreglos por ejemplo)

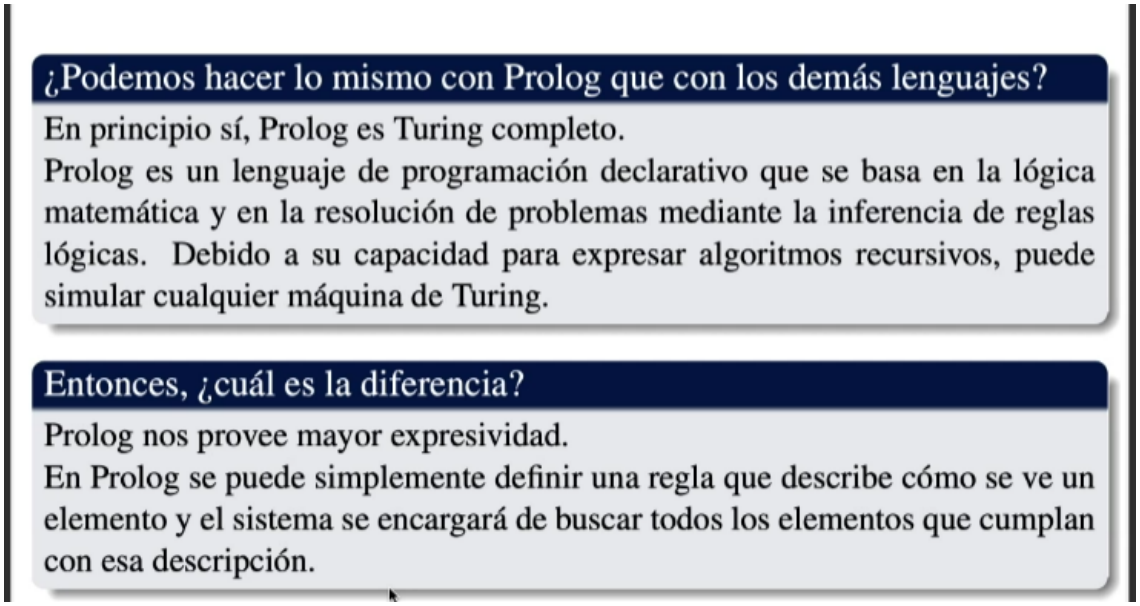
> 💡 Se debe de describir que se quiere hacer no como

Qué tenemos entonces?

- Recursividad
- Listas (no en el sentido C o Lisp)
- Predicados
- MGU (unificador más general). En prolog, el MGU se utiliza para encontrar un sustituto común que permita unificar dos términos y hacer que sean idénticos.
- Motor de inferencia, es el encargado de procesar las reglas y hechos definidos en el programa y encontrar todas las soluciones posibles que satisfagan la consulta del usuario.

Podemos hacer lo mismo con Prolog que con los demás lenguajes?

En principio si, Prolog es Turing completo.

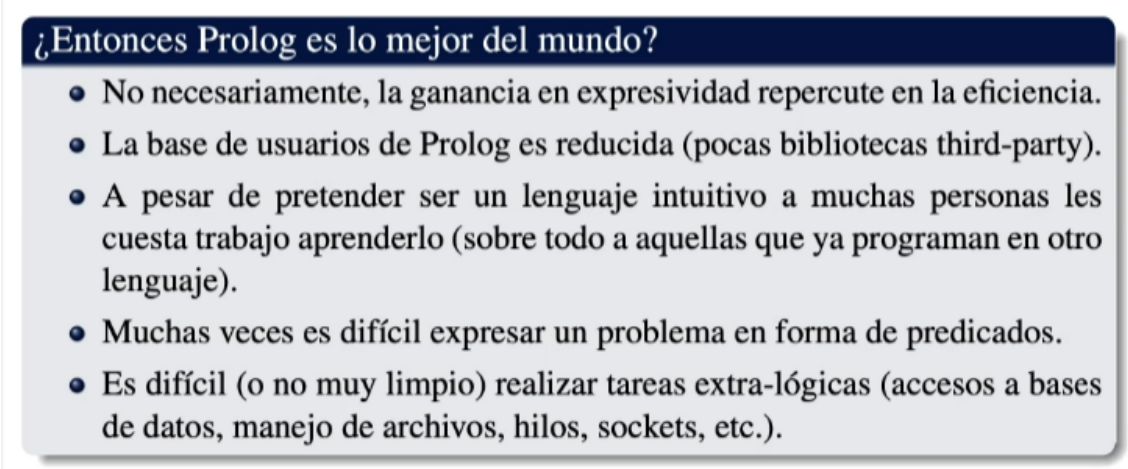


El sistema va a buscar todos los elementos que cumplan con la condición.

Maquina de turing es un modelo lógico

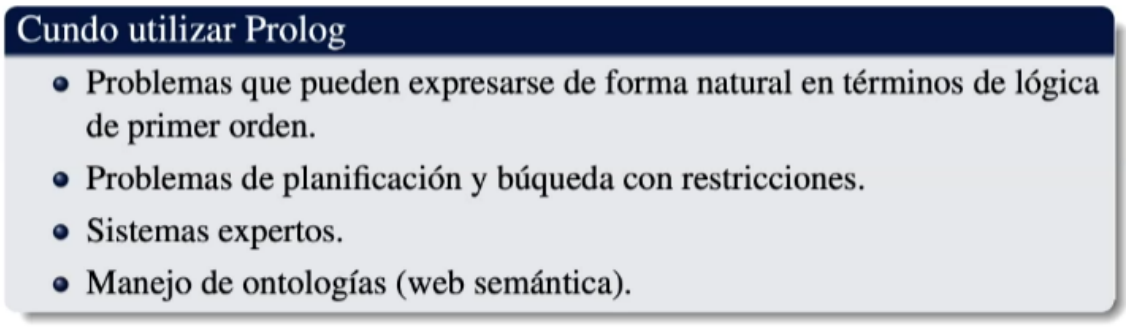
Esto puede evaluar cualquier máquina de Turing

Prolog es lo mejor del mundo?



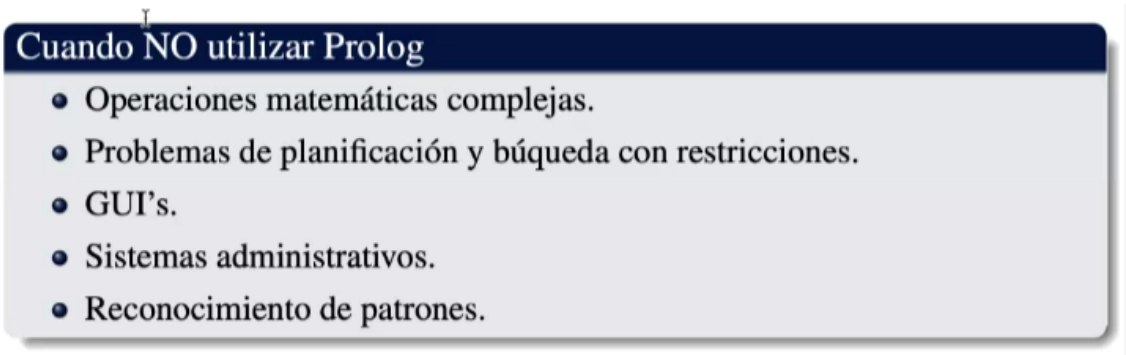
La comunidad de prolog es pequeña.

Cuando utilizar prolog?



Sistemas expertos: Un sistema que tiene cargada una base de conocimiento a partir de un experto que puede dar respuestas como si fuera un experto a quien lo solicite.

Cuando NO utilizar Prolog



Qué necesitamos con Prolog

- Un interprete: STW Prolog
- Un editor: Emacs

Prolog

- El modelo de ejecución en Prolog es muy diferente al de otros lenguajes.
- En Prolog no se sigue el modelo de entrada y salida tradicional.
- En C por ejemplo la idea es crear programas que reciben algo y regresan algo (comportamiento funcional).
- En Prolog la idea es definir una base de conocimientos.
- Dada una base de conocimientos y una pregunta (query) el interprete computa cosas automáticamente utilizando su motor de inferencia.

Lógica de primer orden

Lógica de primer orden

- También llamada lógica de predicados o cálculo de predicados.
- Históricamente desarrollada para tratamientos matemáticos.
- Tiene el poder expresivo suficiente para definir a prácticamente todas las matemáticas.

La lógica de primer orden Tiene 3 cuantificados mas sencilla que las matemáticas habituales para comprobar teoremas, hipotesis, etc para investigaciones.

Cuantificadores

Cuantificadores

- Se caracteriza por el uso de cuantificadores lógicos (para todo $\forall$, existe $\exists$).
- $\forall X$ que sea hombre, X es inteligente.
- $\forall$ indica que todos los elementos de un conjunto dado cumplen con cierta propiedad. Todos los elementos del conjunto hombre cumplen con la propiedad de ser inteligentes.
- $\exists X$ tal que X es mejor a 0.
- $\exists$ indica que al menos un elemento de un conjunto dado cumple con cierta propiedad.

x = hombre

P(x) = Es inteligente

Para todo x que sea hombre es inteligente

Si se genera un conjunto y se define en para todo significa que todos cumplen con la propiedad pero si se pone existe x tal que si es inteligente in se cumple todo.

1. **Cuantificador Universal (∀):**

Supongamos que tenemos el conjunto de números naturales y la propiedad P(x) que significa "x es un número par". La afirmación "∀x P(x)" se lee como "para todo x, x es un número par". Esto significa que todos los números naturales son pares. Esta afirmación es falsa porque los números impares también son parte del conjunto de números naturales.

2. **Cuantificador Existencial (∃):**

Siguiendo con el ejemplo anterior, la afirmación "∃x P(x)" se lee como "existe al menos un x tal que x es un número par". Esto es verdadero, ya que hay al menos un número par en el conjunto de números naturales (por ejemplo, 2).

Para todo numero par o natural x es un numero par, eso significa que todos los numeros naturales son pares

~~Es falso porque los numeros negativos tambien son parte del conjunto de numeros naturales~~

Existe

@18 de abril de 2024

1. Predicados y cuantificadores_ La lógica de primer orden

> 💡 NP - 1 NP - HARD

@19 de abril de 2024

1. Navegación de una proposición atómica

   Si tenemos una proposición atómica **~**P, su negación P Simplemente significa que la proposición P es falsa

2. Navegación de una combinación de proposiciones con conectivos lógicos

   a. La navegación de una conjunción (AND) se convierte en una disyunción (OR) de las negaciones de las proposiciones originales

   > 💡 **~(P ^ Q) = (~P v ~Q)**

   b. La negación de una disyunción (OR) se convierte en una conjunción (AND) de las negaciones de las proposiciones originales

   > 💡 **~(P v Q) = (~P ^ ~Q)**

   c. La negación de una implicación (→) se convierte es un conjunción de la premisa original con la negación de la conjunción

💡 **~(P → Q) = (P ^ ~ Q)**

d. La negación de una equivalencia (↔) se convierte en una disyunción de la conjunción de dos conjunciones

💡 ~P ←→ Q) = (P ^ ~Q) v (~ P ^ Q)

El perro no ladra o no muerde | El perro no ladra ni muerde

Hoy no es un día soleado y no está lloviendo

Estudias y no apruebas el examen

@24 de abril de 2024

Predicados

En prolog se hace referencia a la palabra **predicados**

- Representan relaciones entre los individuos de un mundo
- Ej La tierra es más grande que la luna
- Formalizado: masGrande(Tierra, luna)
- Ej: Pedro es el padre de José

- Formalizado: Padre(Pedro, José)
- Ej: hijo(José, pedro)
- Notar que la semántica de la relación es opaca (la interpretación es abierta.)

Componentes de una base de conocimientos

- Hechos: Predicados, afirmaciones del mundo, Por ejemplo clima(caluroso)
- Los hechos se expresan de la forma: functor(arg1, arg2, ..., arg11). Donde “functor” es una estructura de términos que consta de un nombre y un número fijo de argumentos.
- Reglas: Cláusulas definitivas, afirmaciones del mundo condicionadas por ejemplo: mamifero(x): -pelo(x)
  - La expresión “:-” se utiliza para definir reglas o consultas, se lee como “si” o “implica” por ejemplo:
  - Abuelo(x, y) : - hombre(x), padre(x, z), padre(z, x)

> 💡 la coma “ , “ significa un (Y ｜ &&)

- Las reglas se expresan de la forma:
  - functor(arg1, arg2, ..., arg11) : -
  - functor1(arg1, arg2, ..., arg11) : -
  - functorn(arg1, arg2, ..., arg11) : -

Variables y constantes

- Constantes: Términos en minúsculas, inmutables, no son unificables.
  - El átomo “verde”

        - El npumero negativo .2
        - El átomo “hola”
        - el npumero decimal 3.1416
        - losátomos “true” o “false”
        - “Punto (4, 6)” puede es una constante compuesta que representa el punto X = 4 Y y = 6 EN EL PLANO CARTESIANO
    - Las variables: Terminos cuyo primer carçter es una mayúscula, sin inmutables, son unificables. Por ejemplo, las variables “X” , “Y” , “__” , “Edad” y “nombre Completo” son válidas en prolog

---

- Una clausula lógica es en términos generales una disyunción de literales
- Es una forma particular de representa una regla lógica en la que se establece una premisa (o varias premisas) que implican una conclusión
- En Prolog las cláusulas de Horn se utilizan para definir reglas lógicas que se utilizan en la inferencia
- Por ejemplo abuelo(X,Z):-padre(Y,Z).
- Si la cláusula de Horn tiene una literal positivo se dice que es una cláusula definitiva
- Si la cláusula de Horn no tiene un literal positivo se dice que es un meta (goal o bien query).

Operador =

- Tiene otro significado al de lenguajes como C
- Si una variable no está unificada unifica con lo que esté a la derecha. X = perro.

- Si ya está unificada entonces se comporta como la función de igualdad
- Problema: no siempre se puede saber si la variable ya está unificada
- Solución: Evitar utilizar el operador= , mejor utilizar == para igualdad y el operador is para asignación
- En el caso de asignación, es poco común hacerlas, sólo que estemos trabajando con números.

Interprete

- Al interprete se le plantean preguntas (metas), estas preguntass se realizan sobre una base de conocimientos dada
- El interprete simplemente contesta verdadero o falso:
  - ?clima(templado). R = false
- Se pueden realizar las preguntas de tal modo que es interprete también muestre unificaciones:
  - ?clima(X). R = true.X = Soleado

Operadores relacionales

Operador	Significado	Ejemplo
is	Unificación	X is 10 + 2
=:=	igualdad	10 +2 =;= 2 + 7
=\=	Desigualdad	10 + 2 =\= 5 + 8
>	Mayor que	11 * 3 > 3 ^ 2
<	Menor que	2 ** 10 < 5 * 2
>=	Mayor o igual que	99.0 >= 0
=<	Igual o menor que	-15 =< 15

Operadores relacionales sin evaluación

Operador	Significado	Ejemplo
=	Unificación	X = 10 + 2
==	Igualdad	10 + 2 == 10 + 2
\==	Desigualdad	10 + 2 \== 5 + 7

- Define conceptos en terminos de ellas mismas

- Están definidas por al menos 2 casos
  - Uno terminal no recursivo y una llamada recursiva.

---

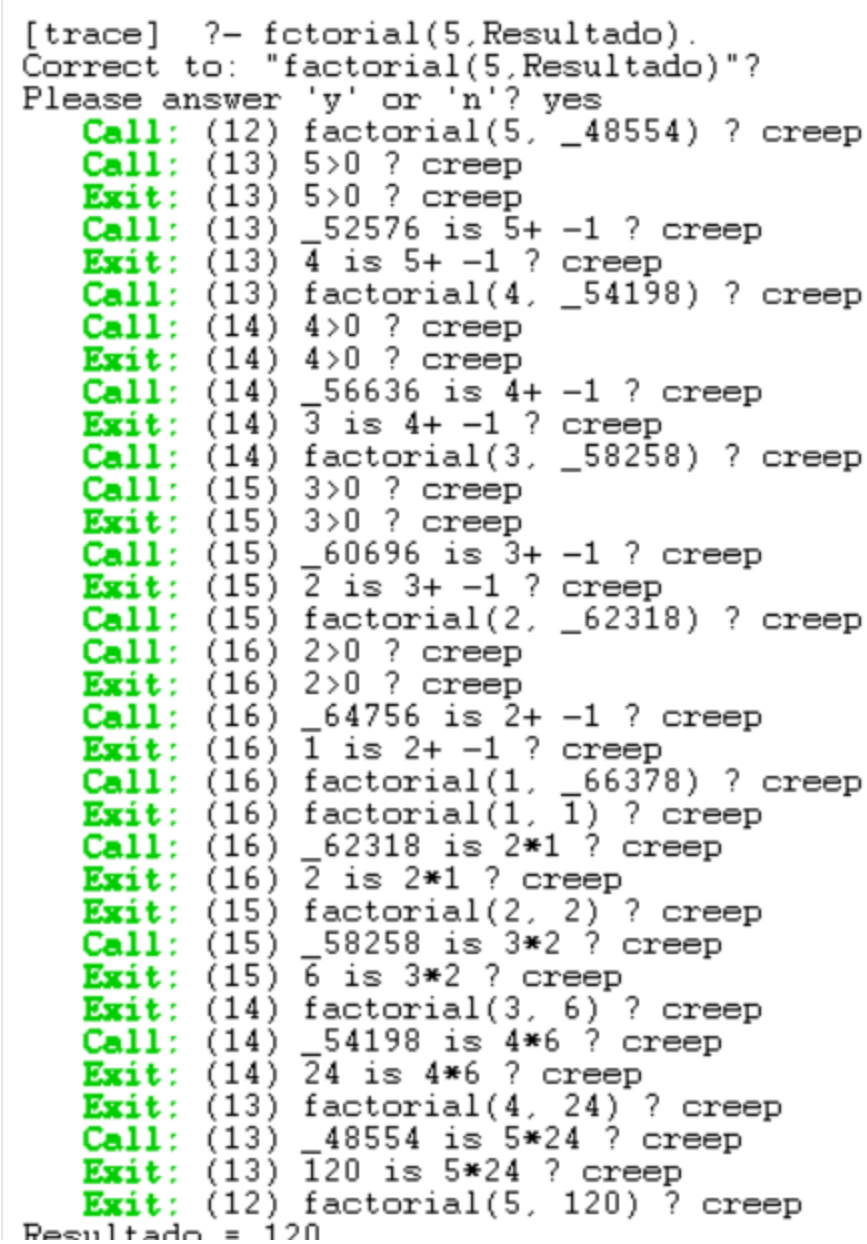
factorial(0, 0).

factorial(1, 1).

factorial es el caso base y (1, 1) los argumentos

```
factorial(X,Y):-
    X > 0,
    X2 is X-1,
    factorial(X2, Y2),
    Y is X * Y2.
```

```
?- consult('C:/Users/hizza/Desktop/4toSemestre/Paradigmas/factorial.pl').
true.

?- consult('C:/Users/hizza/Desktop/4toSemestre/Paradigmas/factorial.pl').
true.

?- factorial(5,Resultado).
Resultado = 120 .

?- trace.
true.

[trace]  ?- factorial(3,Resultado).
   Call: (12) factorial(3, _31162) ? creep
   Call: (13) 3>0 ? creep
   Exit: (13) 3>0 ? creep
   Call: (13) _34088 is 3+ -1 ? creep
   Exit: (13) 2 is 3+ -1 ? creep
   Call: (13) factorial(2, _35710) ? creep
   Call: (14) 2>0 ? creep
   Exit: (14) 2>0 ? creep
   Call: (14) _38148 is 2+ -1 ? creep
   Exit: (14) 1 is 2+ -1 ? creep
   Call: (14) factorial(1, _39770) ? creep
   Exit: (14) factorial(1, 1) ? creep
   Call: (14) _35710 is 2*1 ? creep
   Exit: (14) 2 is 2*1 ? creep
   Exit: (13) factorial(2, 2) ? creep
   Call: (13) _31162 is 3*2 ? creep
   Exit: (13) 6 is 3*2 ? creep
   Exit: (12) factorial(3, 6) ? creep
Resultado = 6 .

[trace]  ?-
```



@9 de mayo de 2024

Motor de inferencia

- El motor de inferencia de Prolog está basado en el algoritmo SLD (Seleccion, Lista y Dife)

- El comportamiento de este motor de inferencia se basa en una búsqueda primero en profundidad
- Cada vez que se le plantea una meta al interprete, este recorre la base de conocimientos de arriba hacia abajo y de izquierda a derecha, este es el orden de las expansiones del árbol.
- Si el motor de inferencia encuentra una hoja que sea exitosa, entonces devolverá True, junto con las unificaciones de variables que permitieron llegar a esa hoja.
- Si llega a una hoja no exitosa entonces regresa un paso y toma la siguiente rama
- Cuando el interprete se le da la orden -tras haber encontrado una posible respuesta: el motor regresa un paso en el árbol e intenta expandirlo hasta otro caso exitoso.

Corte

- Muchas veces no se desea que el motor de inferencia recorta todas las ramas del árbol de búsqueda, sobre todo cuando se desea solo una solución.
- Esto también aumenta la eficiencia de nuestros programas al poderse ignorar ramas fallidas
- Para esto existe el operador de corte “!” de aridad ()
- Solo utilizar el corte en reglas recursivas
- A menos que se está muy seguro de lo que se está haciendo, solo poner el corte sobre un caso terminal
- Poner el corte como última cláusula (si no se pone como última todo lo que le sigue es tratado de forma normal y esto puede tener efectos inesperados).

Variables anonimas

A veces no nos interesa una variable pero debemos proveerla para respetar la aridad de una cláusula.

En estos casos se utiliza “_” en vez de la variable.

Si el interprete da un warning sobre una variable que no estás usando, probablemente puedes utilizar una variable anónima en su lugar.

padre(_,X) Todos los hijos para ese padre

padre(X,_) Todos los padres correspondientes a ese hijo o tantas veces tenga hijos

```
?- imprimir([[1,2,3],[a,b,c],[X,Y,Z]]).
[1,2,3]
[a,b,c]
[_9080,_9086,_9092]
true.

?- imprimir([[1,2,3],[a,b,c],[x,y,z]]).
[1,2,3]
[a,b,c]
[x,y,z]
true.
```

Negación

- Es por fallo finito, esto quiere decir que se busca una contradicción, sino se encuentra entonces la negación es exitosa.
- Su utiliza el predicado por default not
- Tener cuidade al utilizarlo, siempre poner la deficinición del predicado que se quiere negar arriba (en el código del prorama) de la llamada not, sino pueden suceder cosas imprevistas
- Ejemplo

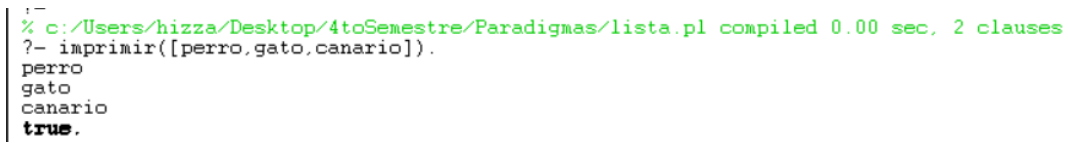
Listas

- Estructura de datos que permite agrupar elementos de manera ordenada
- En prolog las listas pueden contener cero o mas elementos
- Se representan usando corchetes “[ ]”

- Los elementos de las listas se separan por comas
- Las listas pueden contener varios tipos
  - Por ejemplo “[Hola, 1, 2, a, b, 345]”
- Se pueden usar como
  - Argumento de predicados
  - Almacenamiento de datos
  - Estructuras de datos complejas

Elementos de una lista

- Cabeza : primer elemento de la lista
- Cola: resto de la lista (es otra lista)
- Accedemos a los elementos de la lista con el operador “ | ” que se para la cabeza de la cola
- Poe ejemplo:
  - [perro, gato,canario]=[perro|[gato,canario]].
  - Otro ejemplo
  - [[1,2,3],[a,b,c],[X,Y,Z]].



Recorrer elementos de una lista

- El recorrido de cabeza a cola (recorrido secuencial)
- El control del recorrido se logra mediante recursividad
- Normalmente el caso de terminal se representa por la lista vacia

- Por ejemplo
- imprimir([]), - caso base cuando la lista es vacia
- imprimir([H | T]):- print(H), nl, imprimir(T)

Operaciones sobre listas

- Pertenece elemento a lista
- Agregar elemento a la lista
- Concatenar dos listas
- Voltear listas
- Eliminar ocurrencias de un elemento
- Intercalar elementos de dos listas

Listas de listas

- No tienen nada de especial, simplemente cada elemento de la lista es a su vez una lista.
- Los elemntos se tratan de la misma manera.
- A su vez puede haber listas de listas de listas de listas

long([],0).

long([_|Xs],L):- long(Xs,L1),L is L1 + 1.

Recorrido en una lista

%miembro(X,L): El elemento X s miembro de la lista L

miembro(X,[X|_]).

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

Realizar las siguientes consultar y analizar.

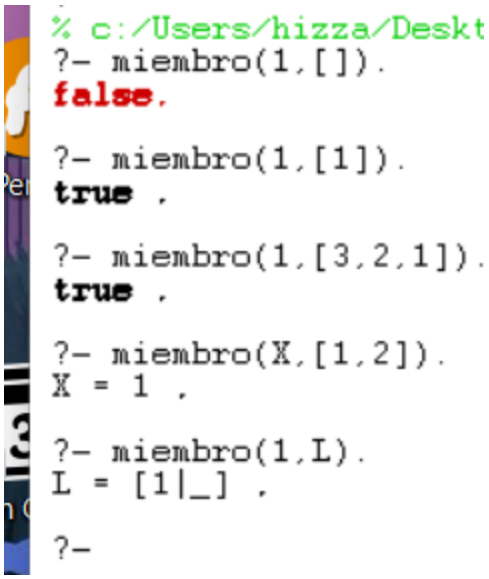
miembro(1,[]).

miembro(1,[1]).

miembro(1,[3,2,1]).

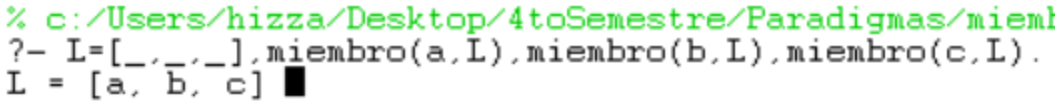
miembro(X,[1,2]).

miembro(1,L).



Ahora prueba con el siguiente ejercicio

L = [_,_,_],miembro(a,L),miembro(b,L),miembro(c,L).



el sistema ha registrado los datos correctamente o excepción el sistema despliega la ventana evidencia registrada

El usuario presiona el botón aceptar de la ventana

El sistema cierra la ventana registrar actividad

FA Documento invalidp

Findall

Se utiliza para encontrar todas las soluciones de una consulta y almacenarlas en una lista. Su sintaxis es:
findall(Template, Goal, List)

- **Template**: Especifica la forma de las soluciones que se recopilarán.
- **Goal**: Especifica la consulta o condición que debe cumplirse para cada solución.
- **List**: Es la lista de todas las instancias de **Template** que satisfacen **Goal**.

Aquí hay algunos ejemplos para ilustrar cómo usar **findall/3**:

```
color(rojo).
color(verde).
color(azul).
color(amarillo).
?- findall(X, color(X), Colores).
```
Colores = [rojo, verde, azul, amarillo].

Bagof

Es similar a findall/3 en que ambas recopilan todas las soluciones de una consulta, pero bagof/3 agrupa las soluciones en listas basadas en valores de variables libres. La sintaxis de bagof/3 es:
bagof(Template, Goal, List)

- **Template**: Especifica la forma de las soluciones que se recopilarán.
- **Goal**: Especifica la consulta o condición que debe cumplirse para cada solución.
- **List**: Es la lista de todas las instancias de **Template** que satisfacen **Goal**.

Una diferencia importante entre findall/3 y bagof/3 es que bagof/3 considera las variables libres (variables que no están especificadas en Goal) y las agrupa en listas separadas según sus valores.

```
persona(juan, 25).
persona(maria, 17).
persona(ana, 30).
persona(pedro, 22).
?- bagof(Nombre, Edad^persona(Nombre, Edad), Lista).
```
Lista = [juan, maria, ana, pedro].

Setof

La función **setof/3** en Prolog es similar a **bagof/3** pero con dos diferencias clave: **setof/3** ordena las soluciones y elimina duplicados. Esto la hace útil cuando necesitas una lista ordenada y sin duplicados de las soluciones que satisfacen una consulta.

setof(Template, Goal, List)

- **Template**: Especifica la forma de las soluciones que se recopilarán.
- **Goal**: Especifica la consulta o condición que debe cumplirse para cada solución.
- **List**: Es la lista ordenada y sin duplicados de todas las instancias de **Template** que satisfacen **Goal**.

```
color(rojo).
color(verde).
color(azul).
color(amarillo).
color(rojo).
```

Solo aparece el resultado una ves independientemente de cuantas veces se repita.

Cuantos hombres de toda la base de conocimiento son papás

Cuantos son abuelo

Cuantos son hijos

Cuantos son mayores de cierta edad

```
persona(juan, 25).
persona(maria, 17).
persona(ana, 30).
persona(pedro, 22).
mayor_de_edad(Nombre) :- persona(Nombre, Edad), Edad > 18.
?- findall(Nombre, mayor_de_edad(Nombre), Mayores).
Mayores = [juan, ana, pedro].
Todos los mayores de edad
¿Cuántos mayores hay?

% Contar el número de personas mayores de 18 años
contar_mayores(Conteo) :-
  findall(Nombre, mayor_de_edad(Nombre), Lista),
  length(Lista, Conteo).

?- contar_mayores(Conteo).
Conteo = 3.
```

Son mayores de 18 y son menores

Ahora, supongamos que queremos agrupar las personas por edades específicas:

```
?- bagof(Nombre, persona(Nombre, Edad), Listas).
Edad = 17,
Listas = [maria] ;
Edad = 22,
Listas = [pedro] ;
Edad = 25,
Listas = [juan] ;
Edad = 30,
Listas = [ana].
```

En este ejemplo, **bagof/3** agrupa las soluciones en listas separadas según el valor de **Edad**. Cada lista contiene las personas que tienen la misma edad.