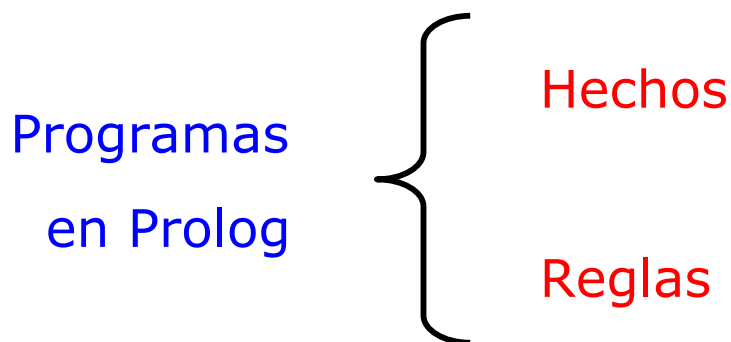


INTRODUCCIÓN A PROLOG

- Lenguaje declarativo (opuesto a procedimentales) basado en reglas de la lógica
- PROLOG = "Programming in Logic"
- Originado en Europa a principios de los 70's por Alain Colmerauer (Universidad de Marsella, Francia)



- La información se extrae por medio de *consultas*

HECHOS

- Propositiones:
 - "Juan es un programador"
 - "El león es un mamífero"

```
programador(juan).  
mamifero(leon).
```

- Nótese que se anota primero el *predicado* y entre paréntesis el sujeto de la proposición

SINTAXIS

- Variables: Mayúsculas.
- Constantes: Minúsculas.
- Cada afirmación debe terminar con un punto.
- No se aceptan constantes con espacios en blanco, en lugar de ello se usa el

guión bajo (_). P. ejem.
`primer_ministro.`

- Los comentarios empiezan con el símbolo **%**

RELACIONES

- “Juan es el padre de Miguel”

`padre(juan, miguel).`

REGLAS

- Sentencias condicionales
- “Si el león come carne, entonces es carnívoro”

`carnivoro(leon):- comecarne(leon)`

CONJUNCIONES

- Emplea el operador lógico AND
- Se utiliza la coma (,)

`tia(X,Y):-hermana(X,Z),padre(Z,Y).`



Regla
(Condición)



AND

DISYUNCIONES

- Emplea el operador lógico OR
- Se utiliza el punto y coma (;)

`hijo(X,Y):-padre(Y,X);madre(Y,X).`



Regla
(Condición)



OR

CONSULTAS

- No es necesario programar el mecanismo de búsqueda
- Utiliza la *resolución* en sus derivaciones (generalización del *modus ponendo ponens* junto con la *unificación*)
- P. ejem.

```
lagarto(iguana).  
serpiente(vibora).  
mamifero(conejo).
```

Las consultas pueden ser interactivas mediante el indicador de comandos (**?-**):

?- lagarto(iguana). ¿Una iguana es un lagarto? (Termine la expresión con punto)

Yes.

Respuesta de Prolog

?-	Indicador que espera el sig. Comando
?-serpiente(conejo).	¿Un conejo es serpiente?
no.	Respuesta de Prolog

Se pueden usar variables en las consultas:

?- lagarto(X).	Nombre de un lagarto (Note la X mayúscula)
X=iguana	Respuesta de Prolog
Yes	(Oprima ENTER para terminar la consulta o ESPACIO para buscar otra ocurrencia)

Otro ejemplo:

```
pais(usa).  
pais(canada).  
pais(mexico).  
capital(usa,washington).  
capital(canada,ottawa).  
capital(mexico,cd_mexico).
```

Consultas realizadas:

?- pais(mexico).
yes.

?- capital(canada,washington).
no.

?- pais(japon).
no.

?- pais(X).
X=usa
X=canada
X=mexico
yes.

?- capital(canada,B).

B=ottawa

yes.

?- capital(R,washington).

R=usa

yes.

BACKTRACKING (RETROCESO)

- Las consultas pueden tener una o varias metas
- Consideremos el sig. ejemplo:

`compra(X,Y):-sevende(Y),gusta(X,Y),bueno(Y).`

`sevende(vestido).`
`sevende(sombrero).`
`sevende(zapatos).`
`gusta(jaime,zapatos).`
`gusta(maria,vestido).`
`gusta(maria,sombrero).`
`bueno(sombrero).`

- La regla *compra* tiene éxito si todas sus metas tienen éxito.

- Prolog intenta satisfacer las metas de la consulta de izquierda a derecha y para cada meta va probando las cláusulas correspondientes

1er. Intento:

```
compra(Z,vestido):-  
sevende(vestido),gusta(Z,vestido),bueno  
(vestido).
```

- Busca la cláusula para sustituir **Z** (de izq. a der.), teniendo a **vestido** como segundo argumento y encuentra `gusta(maria,vestido)`
- Se tiene ahora:

```
compra(maria,vestido):-  
sevende(vestido),gusta(maria,vestido),  
bueno(vestido).
```

- La regla fracasa porque no hay regla que satisfaga `bueno(vestido)`

- Esto no significa que `compra(maria, vestido)` haya fracasado, sino que se ha seleccionado una cláusula que no conduce a la solución.
- Por esa razón es necesario aplicar un *retroceso* (*backtracking*).

2o. Intento:

```
compra(Z, sombrero) :-  
  sevende(sombrero, gusta(Z, sombrero),  
          bueno(sombrero)).
```

- Busca la cláusula para sustituir **Z** (de izq. a der.), teniendo a **vestido** como segundo argumento y encuentra `gusta(maria, sombrero)`

- Se tiene ahora:

```
compra(maria,sombrero):-  
sevende(sombrero),gusta(maria,sombrero,  
bueno(sombrero)).
```

- La regla tiene éxito porque se hace una prueba satisfactoria a `bueno(sombrero)`

ENTRADA Y SALIDA

- Se usa el comando `write` para desplegar un texto o una variable en la pantalla

```
write('Hola...').
```

```
saludo:- nl, tab(4),write('Hola '),  
nl, tab(20), write(X).
```

- Se usa el comando `read` para capturar desde el teclado

```
name:- write('Anote su nombre:'),
        read(Nombre), nl, write('Hola
                                '),write(Nombre).
```

- Nótese que la variable `Nombre` inicia con mayúscula.

ESTRUCTURAS

- Se pueden utilizar varios datos a la vez:
`nacimiento(pedro, fecha(23, ago, 1970))`.

?- `nacimiento(pedro, X)`.
`X=fecha(23, ago, 1970)`
yes.

- Consultas de todas las personas nacidas en Agosto:

?- nacimiento(X, fecha(Y, ago, Z)).

X=pedro

yes.

ARITMÉTICA

- Se usa el predicado `is`

?- X is 3+4

X=7

yes.

- Uso de operaciones aritméticas en predicados:

`suma(A, B, C) :- C is A + B.`

?- suma(3, 4, 7).

yes.

?- suma(3, 4, X).

X=7

yes.

CICLOS

- En Prolog, casi no se usan ciclos, en lugar de ellos se aplica recursividad; sin embargo, se pueden implementar.
- P. ejem. Para imprimir los números del 1 al 10 se usa ...

```
lista(M, N):- M<N, nl, write(M),  
NuevoM is M+1, lista(NuevoM, N).
```

LISTAS

- En Prolog no hay matrices, en su lugar se usan Listas.

```
[maria, javier, juan]  
[] %lista vacía
```

CABEZA Y COLA DE LISTAS

- Si se tiene la lista `[a, b, c, d]`, la `a` es la cabeza y la cola es la lista `[b, c, d]`
- Una lista cuya cabeza es `A` y cola es `B` se anota como `[A | B]`
- El predicado

`primer_elemento(X, [X|_]).`

tiene éxito si `X` es el primer elemento de la lista.

IMPRIMIR LOS ELEMENTOS DE UNA LISTA

- Si la lista no está vacía, primero se imprime la cabeza y luego la cola:

```
imprimir( [A | B] ):- write(A),  
                    imprimir(B).
```

AGREGAR ELEMENTOS A UNA LISTA

- Este predicado tiene tres listas: A, B y C, donde A y B se fusionan en el mismo orden y generan C:

```
agregar([A | ColaA], B, [A, ColaC]):-  
        agregar(ColaA, B, ColaC).
```


LA NEGACIÓN COMO FRACASO

- El predicado `\+` tiene éxito sólo si fracasa su argumento.
- Considere los siguientes hechos:

`roja(rosa).`

`verde(hierba).`

`blanca(margarita).`

- Suponga la siguiente consulta:

`?- roja(amapola).`

`no.`

- Esto no significa que las amapolas no sean rojas, sino que no hay hechos que lo confirmen. O sea que

`?- \+roja(amapola).`

`yes.`

`?- \+roja(rosa).`

`no.`

CORTES

- Es un predicado que cuando se invoca detiene las inferencias y fija las decisiones tomadas hasta ese momento.
- Impide el retroceso
- Se denota por el símbolo **!**
- No se puede rehacer ninguna meta que precede al corte
- Sólo se permite el retroceso a las metas ubicadas después del corte

```
melodia(X) :- la(X),!, re(X), mi(X).
```

- Una vez que `la(X)` tiene éxito, se hace el corte y esto fija todas las opciones.
- No se puede rehacer `la(X)`

- Cualquier otra meta que tenga el predicado `melodia(X)` se excluirá de las consideraciones posteriores

LAS LISTAS COMO EXPRESIONES LÓGICAS

- Representación de operadores lógicos como listas
- La *negación* se puede representar como una lista de dos elementos: la palabra `neg` y una *expresión lógica*.
- Las *conjunciones*, *disyunciones*, *implicaciones* y *equivalencias* se pueden representar como una lista de tres

elementos: el alcance izquierdo, la conexión lógica y el alcance derecho.

- Definiciones:

1. Conexiones binarias	and, or, then, iif
2. Constantes lógicas	v y f
3. Variables lógicas	Son expresiones
4. Si A es una expresión	[neg, A] también lo es
5. Si A y B son expresiones y Con es una conexión lógica	[A, Con, B] es una expresión

- Dichas definiciones se pueden traducir a Prolog y hacerse ejecutables.
- Se abrevian constantes lógicas (`clog`), variables lógicas (`vlog`), conexión binaria (`conbi`) y expresiones (`expr`).
- Se supone que las variables lógicas son **p, q, r y s**.
- El programa queda de la sig. forma:

```
conbi (and) .
conbi (or) .
conbi (then) .
conbi (iif) .
clog (v) .
clog (f) .
vlog (p) .
vlog (q) .
vlog (r) .
vlog (s) .
expr (X) :-clog (X) .
expr (X) :-vlog (X) .
expr ( [neg, A] ) :-expr (A) .
expr ( [A, Con, B] ) :-expr (A) ,
conbi (Con) , expr (B) .
```

- Para probar las expresiones y verificar que estén bien formadas:

$p \& (q \rightarrow \neg r)$

?- expr([p, and, [q, then, [neg, r]]]).
yes.

EVALUACIÓN DE EXPRESIONES LÓGICAS

- Se supone que todo aquello que no se pueda demostrar que es cierto, se considera falso por definición.

- Consideraciones:
 1. La constante lógica \forall es verdadera
 2. Existe una lista de variables lógicas, todas las cuales son ciertas por definición.
 3. $[\text{neg}, A]$ es cierto si A es falsa.
 4. $[A, \text{and}, B]$ es cierto si A es cierto y B es cierto.
 5. $[A, \text{or}, B]$ es cierto si A es cierto.
 6. $[A, \text{or}, B]$ es cierto si B es cierto.
 7. $[A, \text{then}, B]$ es cierto si A no es cierto.
 8. $[A, \text{then}, B]$ es cierto si B es cierto.
 9. $[A, \text{iif}, B]$ es cierto si $[A, \text{then}, B]$ y $[B, \text{then}, A]$ son ciertos.

- Suponemos que q y r son ciertas y todas las demás variables son falsas:

```
true(q).  
true(r).  
true([neg, A]):-\+(true(A)).  
true([A, and, B]):-true(A),  
true(B).  
true([A, or, B]):-true(A).  
true([A, or, B]):-true(B).  
true([A, then, B]):-true(B).  
true([A, then, B]):-\+(true(A)).  
true([A, iif, B]):-true(  
[A, then, B]), true([B, then, A]).
```


EL PREDICADO ASSERT

- Este predicado toma un argumento que debe ser instanciado a una cláusula.
- Agrega un hecho a la base de conocimiento.
- Tiene dos variantes:
 - `asserta(X)` coloca la cláusula instanciada a `X` **antes** de otra cláusula del mismo predicado
 - `assertz(X)` coloca la cláusula instanciada a `X` **después** de otra cláusula del mismo predicado

?- `asserta(hombre(juan))`.
yes.

?- `hombre(X)`.
X=juan.

EL PREDICADO RETRACT

- Este predicado toma un argumento que debe ser eliminado de una cláusula.
- Elimina un hecho de la base de conocimiento.

?- **asserta(hombre(juan))**.
yes.

?- **asserta(hombre(pedro))**.
yes.

?- **hombre(X)**.
X=juan;
X=pedro
yes.

?- **retract(hombre(juan))**.
yes.

?- **hombre(X)**.
X=pedro
yes.

EJERCICIO

- Elabore un programa en Prolog con su árbol genealógico, donde los hechos sean únicamente predicados del tipo padre (- , -) ó madre (-, -)
- Programe los predicados con las reglas necesarias para encontrar las relaciones de parentesco más comunes, tales como:
 - a) hermano (A, B) .
 - b) primo (A, B) .
 - c) tío (A, B) .
 - d) hijo (A, B) .
 - e) nieto (A, B) .
 - f) abuelo (A, B) .
 - g) bisabuelo (A, B) .
 - h) bisnieto (A, B) .
 - i) cuñado (A, B) .
 - j) concuño (A, B) .
 - k) etc.